МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ:

Система розпізнавання та перекладу жестової мови

Галузь знань **12 «Інформаційні технології»**

Спеціальність **122 «Комп'ютерні науки»**

Освітня програма «**Комп'ютерні науки»**

Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи КН- 41,
Корнієнко Д.О.

Керівник Циганок В. В. д.т.н., с.н.с.

Випускна кваліфікаційна робота бакалавра допущена до захисту рішенням кафедри *інтелектуальних технологій*
Протокол №_13__від_05.06.2023 р.

зав. кафедри_____доц. Іларіонов О.Є.

Київ – 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет інформаційних технологій

Кафедра інтелектуальних технологій

Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри
інтелектуальних технологій
Іларіонов О.Є.

_____

"___"_____2023 р.

ЗАВДАННЯ

НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

_____Корнієнкові Денису Олександровичі_____

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)

Система розпізнавання та перекладу жестової мови _____
_____
_____

затверджена протоколом засідання кафедри від « 11 » листопада 2022 р. протокол №4

2. Термін здачі студентом закінченого проекту (роботи) 29 травня 2023 року

3. Вихідні дані до проекту (роботи)

_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

Вступ, особливості задачі розпізнавання жестів та відеозв'язку через інтернет, проектні рішення, архітектура додатка та інтеграція потрібних технологій, реалізація мобільного додатку месенджера та модуля навчання нейромережі розпізнавання жестів, висновки._____

5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)

Вступ. Дані ВООЗ (1-2 слайд), Бар'єр у спілкуванні (3 слайд), Мета роботи (4 слайд), Поняття жестової мови (5 слайд), Аналіз алгоритмів розпізнавання образів (6 слайд), Нейромережа SSD (7-9 сдайди), Протокол WebRTC (10 слайд), Технології (11 слайд), MVP (12 слайд), Концепція. Компоненти (13

слайд), Концепція. Діагарама діяльності системи (14 слайд) , Побудова власного датасету (15 слайд), Демонстрація розпізнавання образів (16 слайд), Побудова мобільного додатка (17 слайд), Демонстрація роботи мобільного додатка (18 слайд), Висновки (19 слайд).

6. Консультанти з випускної кваліфікаційної роботи із зазначенням розділів випускної кваліфікаційної роботи, що їх стосуються

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Перший	Циганок В. В.	15.02.2023	15.02.2023
Другий	Циганок В. В.	14.03.2023	14.03.2023
Третій	Циганок В. В.	7.04.2023	7.04.2023

7. Дата видачі завдання 15 лютого 2022 року

Керівник _____ / Циганок Віталій Володимирович/
(підпис) (ПІБ)

Завдання прийняв до виконання _____ / Корнієнко Денис Олександрович /
(підпис) (ПІБ)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів випускної кваліфікаційної роботи	Примітка
1	Перший розділ. ОСОБЛИВОСТІ ЗАДАЧІ РОЗПІЗНАВАННЯ ЖЕСТІВ ТА ВІДЕОЗВ'ЯЗКУ ЧЕРЕЗ ІНТЕРНЕТ	01.03.2023р.	
2	Другий розділ. ПРОЕКТНІ РІШЕННЯ. АРХІТЕКТУРА ДОДАТКА ТА ІНТЕГРАЦІЯ ПОТРІБНИХ ТЕХНОЛОГІЙ	10.04.2023р.	
3	Третій розділ. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ МЕСЕНДЖЕРА ТА МОДУЛЯ НАВЧАННЯ НЕЙРОМЕРЕЖІ РОЗПІЗНАВАННЯ ЖЕСТІВ	01.05.2023р	
4	Висновки	10.05.2023р	
5	Виконання презентаційного матеріалу	20.05.2023р	
6	Передзахист	30.05.2023р	
7	Здача готової дипломної роботи на кафедру.	07.06.2023р	

Студент-дипломник _____ / Корнієнко Денис Олександрович /
(підпис) (ПІБ)

Керівник випускної кваліфікаційної роботи _____ / Циганок Віталій Володимирович /
(підпис) (ПІБ)

АНОТАЦІЯ

Корнієнко Денис Олександрович виконав випускну кваліфікаційну роботу на тему «Система розпізнавання та перекладу жестової мови» за спеціальністю 122 – «Комп’ютерні науки».

У випускній кваліфікаційній роботі проведено аналіз жестів української мови глухонімих, сучасних алгоритмів побудови нейромереж розпізнавання образів та протоколи встановлення відеозв’язку. Розглянуті сучасні додатки для спілкування. Розроблено програмне забезпечення для розпізнавання літер абетки дактиля та додаток-месенджер для телекомунікації.

Ключові слова: розпізнавання жестів, дактиль, месенджер, відеозв’язок.

SUMMARY

The degree project: «Sign language recognition and translation system» has completed by Denys Korniienko specialty 122 – «Computer Scienses».

The final qualification work analyzes the gestures of the Ukrainian language of the deaf and dumb, modern algorithms for building pattern recognition neural networks, and protocols for establishing video communication. Modern applications for communication are considered. Software for recognizing the letters of the dactyl alphabet and a messenger application for telecommunications have been developed.

Keywords: gesture recognition, dactyl, messenger, video communication

ЗМІСТ

ВСТУП

Метою роботи є розробка додатку-месенджера, що дозволить здійснювати відеодзвінки та листування для забезпечення комунікації між здоровими людьми та представниками глухонімих без необхідності знання жестової мови. За офіційними даними Всесвітньої організації охорони здоров'я (ВООЗ), глухота та втрата слуху є одними з найпоширеніших проблем в світі. Ці проблеми зустрічаються у кожному регіоні та країні світу, і мають значний вплив на якість життя людей. За даними ВООЗ, понад 1,5 мільярда людей по всьому світу, або майже 20% населення планети, живуть із втратою слуху. Це означає, що більш як одна п'ята частина населення світу стикається з цими проблемами щодня.

Для багатьох людей із вадами слуху, основним способом спілкування є жестова мова. Понад 70 мільйонів глухих людей по всьому світу використовують жестову мову, щоб спілкуватися між собою. Але, на жаль, не всі здорові люди знають жестову мову, що робить комунікацію між здоровими людьми і представниками глухонімих досить важкою. На практиці, не всі люди з вадами слуху знають як спілкуватися в письмовій формі через складність навчання. Це може створювати бар’єри у спілкуванні та унеможливлювати повноцінне спілкування.

В розробленому додатку буде використовуватися технології машинного навчання для розпізнавання жестової мови та перекладу звуку на текст. Користувач може говорити в мікрофон, а потім побачити транскрипцію своєї мови на екрані телефону. При цьому, додаток автоматично буде перекладати текст на жестову мову, яку можна відтворити на екрані. І навпаки, жестова мова буде перекладатися в текст, який здорова людина побачить на екрані. Зв'язок між користувачами буде забезпечуватися за допомогою відео-дзвінків, які є інтуїтивно зрозумілими та зручними у використанні.

Розроблений додаток має потенціал стати цінним інструментом у багатьох сферах життя. Наприклад, в освіті додаток зможе бути використаним для спілкування вчителя та учня з вадами слуху. У бізнесі, додаток може

допомогти спілкуватися з колегами, клієнтами та партнерами, які мають втрату слуху. Також, додаток може бути корисним у повсякденному житті, де він може допомогти рідним та друзям глухонімих людей з легкістю спілкуватися з ними.

Важливо зазначити, що розроблений додаток не замінює жестову мову, а доповнює її. Додаток надає можливість глухонімим людям легко спілкуватися з тими, хто не знає жестової мови, що допомагає зменшити бар'єри в спілкуванні між здоровими людьми та людьми з вадами слуху. Додаток також може зробити спілкування з глухонімими людьми більш доступним та зручним для всіх.

Об'єкт дослідження: жестова мова та її розпізнавання.
Предмет дослідження: згорткова нейронна мережа SSD для розпізнавання образів (жестів) та інтернет-протокол WebRTC для організації голосового та відеозв'язку через інтернет у режимі реального часу.

Задачі дослідження:

1. Визначити сутність жестової мови.
2. Провезти аналіз основних алгоритмів розпізнавання образів.
3. Виявити специфіку навчання нейромереж розпізнавання образів.
4. Проаналізувати протокол WebRTC та його реалізацію за допомогою мов програмування.
5. Створити власну модель розпізнавання жестів.
6. Створити додаток, який реалізує відеозв’язок та переклад мови жестів.

РОЗДІЛ 1 ОСОБЛИВОСТІ ЗАДАЧІ РОЗПІЗНАВАННЯ ЖЕСТІВ ТА ВІДЕОЗВ'ЯЗКУ ЧЕРЕЗ ІНТЕРНЕТ

1.1 Вимоги до системи розпізнавання жестів та відеозв'язку

Функціональні вимоги:

1. Додаток повинен забезпечувати встановлення зв'язку між здоровою людиною та глухонімою у легкий спосіб шляхом відео-дзвінка.
2. Додаток повинен мати можливість перекладати звичайну мову (звук або текст) на мову жестів, та навпаки (мову жестів перекладати на звичайну мову у вигляді субтитрів або синтезу голоса) в режимі реального часу.

Нефункціональні вимоги:

1. Додаток повинен бути доступним для встановлення на пристрої з операційною системою Android та iOS.
2. Додаток повинен мати зручний і інтуїтивно зрозумілий інтерфейс користувача.
3. Додаток повинен працювати стабільно та швидко, навіть при обміні повідомленнями з великим обсягом тексту.
4. Додаток повинен забезпечувати безпеку відео-дзвінків та захист особистих даних користувачів.
5. Додаток повинен мати можливість працювати в офлайн-режимі та забезпечувати синхронізацію повідомлень після підключення до Інтернету.

Вхідною інформацією для додатку є:

1. Користувачів: відомості про користувачів, такі як ім'я, електронна адреса та пароль, необхідні для реєстрації в системі.
2. Запити на встановлення зв'язку: запити від користувачів, що хочуть встановити зв'язок з глухонімим.
3. Відео-з'єднання: інформація, що передається під час відео-з'єднання між здоровим користувачем та глухонімим.
4. Повідомлення: текстові повідомлення, які користувачі можуть відправляти один одному.
5. Повідомлення про помилки: інформація про помилки, що виникають під час роботи додатку.

6. Інші дані: додаткова інформація, яка може знадобитися для забезпечення коректної роботи додатку, така як конфігураційні файли та дані про налаштування користувачів.

Основною вихідною інформацією є успішне встановлення зв'язку між користувачами додатку-месенджера. Завдяки цьому користувачі зможуть спілкуватися між собою за допомогою відео-дзвінків без необхідності знання мови жестів. Додаток повинен забезпечувати якісне передавання звуку і зображення в режимі реального часу.

Крім того, додаток повинен забезпечувати можливість збереження історії розмов, щоб користувачі могли переглядати їх у майбутньому. Вихідна інформація також може включати повідомлення про стан підключення, якщо під час спілкування виникнуть проблеми зі з'єднанням.

1.2 Поняття жестової мови

Жестова мова, рідше мову жестів - самостійний, природно виник або штучно створений мова, що складається з комбінації жестів, кожен з яких виробляється руками в поєднанні з мімікою, формою або рухом рота і губ, а також у поєднанні з положенням корпусу тіла [1]. Ці мови в основному використовуються в культурі глухих з метою комунікації. Використання жестових мов людьми без порушення слуху вдруге, проте досить поширене.

Слова жестових мов, як і слова звичайних мов, складаються з елементарних, що не володіють сенсом компонентів - хірем (аналогія в звичайних мовах - фонеми). Жест може складатися з 5 елементів, об'єднаних в акронім HOLME за першими літерами англійських слів:

· Handshape (форма руки),

· Orientation (орієнтація долоні),

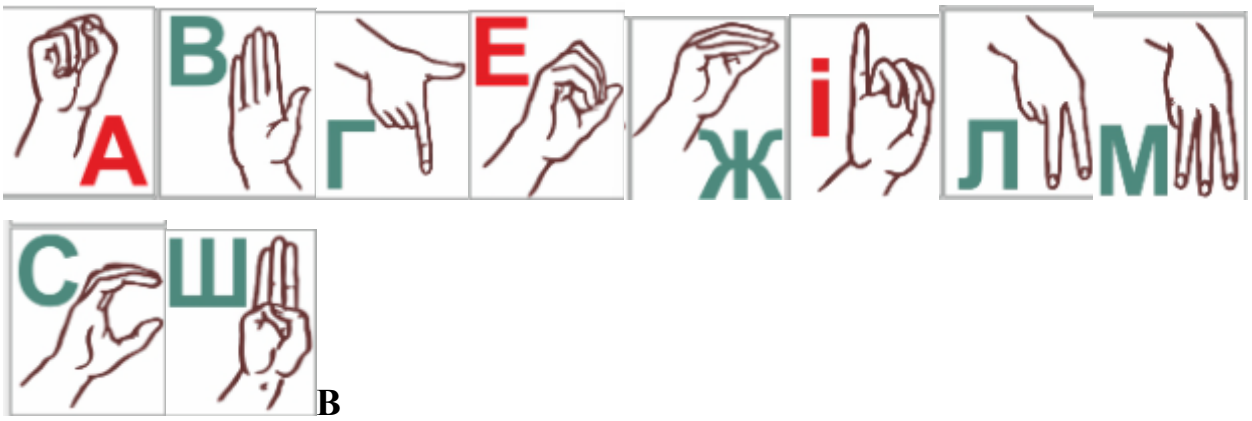
· Location (місце розташування руки, артикуляція),

· Movement (рух),

· Facial Expression (міміка, іноді: неручной маркери).

1.3 Як вивчається жестова мова

Спочатку вивчається дактиль, тобто, абетка (пальці, складені певним чином, позначають одну літеру).



В

Рисунок 1.1 – Деякі букви дактиля [2]

Далі вивчаються слова. Кожна країна має свою жестову мову, а кожне місто — ще і свій сленг, якісь особливі жести, зрозумілі лише цьому конкретному регіону

Рисунок 1.2 – Жест "Привіт" [3]

Рисунок 1.3 – Жест "Вибач" [4]

Рисунок 1.4 – Жест "Так" [5]

1.4 Аналіз існуючих рішень відеозв’язку

Відеотелефонія, також відома як відеоконференції та відеотелеконференції, — це двосторонній або багатоточковий прийом і передача аудіо- та відеосигналів людьми в різних місцях для спілкування в реальному часі. Відеотелефон — це телефон з відеокамерою та відеодисплеєм, здатний одночасно здійснювати відео- та аудіозв'язок.

На 2023 рік існують додатки месенджери, які підтримують відеодзвінки через інтернет (Telegram, Viber, WhatsApp), а також додатки для відеоконференцій (Zoom, Teems, Discord).

Telegram — це глобально доступна безкоштовна, кросплатформна, зашифрована, хмарна та централізована служба обміну миттєвими повідомленнями . Програма також надає додаткові наскрізні зашифровані чати, широко відомі як секретний чат і відеодзвінки, VoIP, обмін файлами та кілька інших функцій.

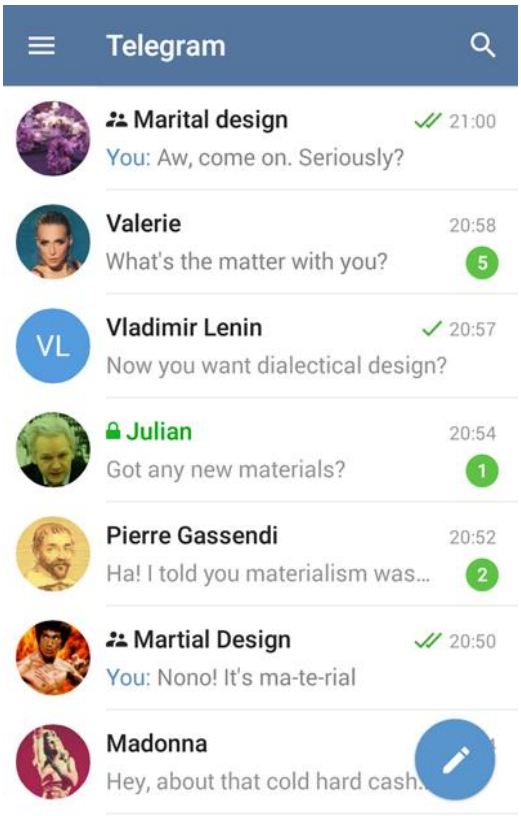


Рисунок 1.5 – Інтерфейс «Telegram» [6]

WhatsApp (також званий WhatsApp Messenger) — це міжнародно доступна безкоштовна міжплатформна служба централізованого обміну миттєвими повідомленнями (IM) і голосового зв’язку (VoIP), що належить американському технологічному конгломерату Meta. Це дозволяє користувачам надсилати текстові та голосові повідомлення, здійснювати голосові та відеодзвінки, а також обмінюватися зображеннями, документами, місцем розташування користувачів та іншим вмістом. Клієнтська програма WhatsApp працює на мобільних пристроях і доступна з комп’ютера.

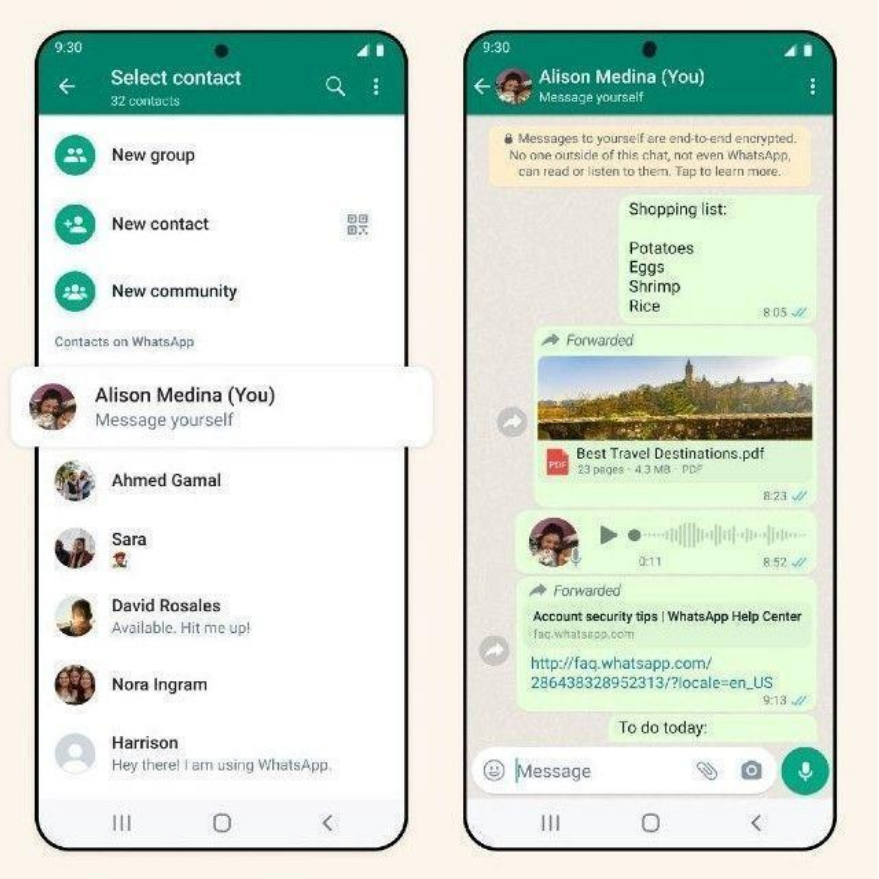


Рисунок 1.6 – Інтерфейс «WhatsApp» [7]

Zoom - це запатентована програма для відеотелефонії, розроблена компанією Zoom Video Communications. Безкоштовний план дозволяє до 100 одночасних учасників із 40-хвилинним обмеженням часу. Користувачі мають можливість оновити, підписавшись на платний план. Найвищий план підтримує до 1000 одночасних учасників для зустрічей тривалістю до 30 годин.

Під час пандемії COVID-19 спостерігалося значне зростання використання Zoom для віддаленої роботи, дистанційної освіти та соціальних зв’язків в Інтернеті. Це збільшення призвело до того, що у 2020 році Zoom став одним із найбільш завантажуваних мобільних додатків у світі з понад 500 мільйонами завантажень і понад 300 мільйонами учасників щоденних зустрічей.

Рисунок 1.7 – Інтерфейс «Zoom» [8]

1.5 Аналіз алгоритмів розпізнавання образів

На практиці існує два типи основних алгоритмів виявлення об'єктів. *R-CNN і Fast(er) R-CNN* використовують двоетапний підхід: спочатку визначають регіони, де очікується виявлення об’єктів, а потім виявляють об’єкти лише в цих регіонах за допомогою згорткової нейроної мережі [7].

YOLO (You Only Look Once) і SSD (Single-Shot Detector), використовують повністю згортковий підхід, за якого мережа здатна знайти всі об’єкти на зображенні за один прохід через мережу.

Single-Shot Detector (SSD) складається з двох компонентів: опорної моделі та голови SSD [8]. Базовою моделлю зазвичай є попередньо навчена мережа класифікації зображень як екстрактор ознак. Зазвичай це мережа на кшталт ResNet, навчена на ImageNet, з якої було видалено остаточний повністю підключений рівень класифікації. Таким чином, у нас залишається глибока нейронна мережа, яка здатна витягувати семантичне значення із вхідного

зображення, зберігаючи при цьому просторову структуру зображення, хоча й із меншою роздільною здатністю. Голова SSD — це лише один або кілька згорткових шарів, доданих до цієї основи, а виходи інтерпретуються як обмежувальні прямокутники та класи об’єктів у просторовому розташуванні активацій кінцевих шарів.

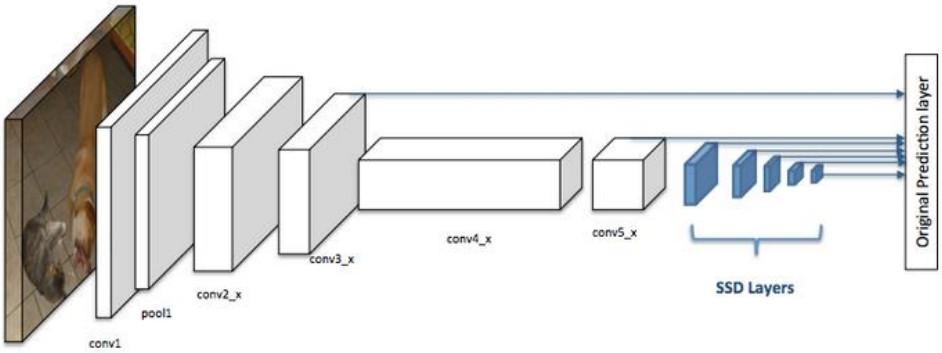


Рисунок 1.8 – Архітектура згорткової нейронної мережі з детектором SSD [9]

SSD ділить зображення за допомогою сітки, і кожна клітинка сітки відповідає за виявлення об’єктів у цій області зображення. Виявлення об’єктів означає просто передбачення класу та розташування об’єкта в цьому регіоні. Якщо жодного об’єкта немає, ми розглядаємо його як фоновий клас, а розташування ігноруємо. Наприклад, ми можемо використовувати сітку 4x4 у прикладі нижче. Кожна клітинка сітки може виводити положення та форму об’єкта, який вона містить.

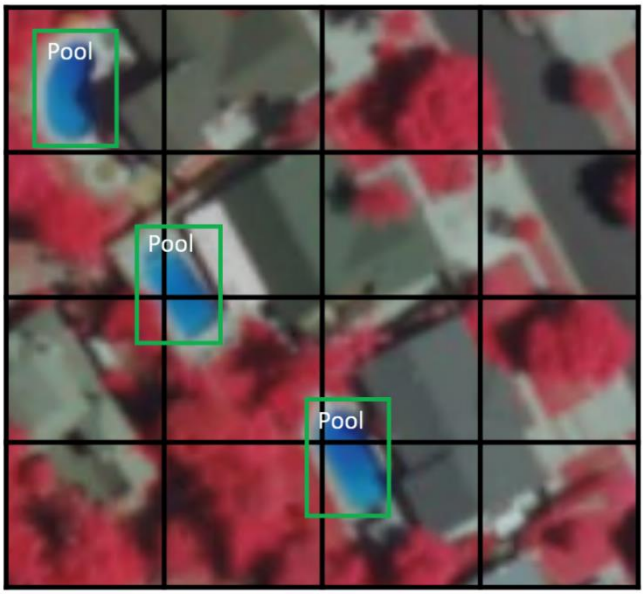


Рисунок 1.9 – Приклад сітки 4x4 [9]

Кожній клітинці сітки в SSD можна призначити кілька прив’язок (прямокутників). Ці прив’язки попередньо визначені, і кожна з них відповідає за розмір і форму в клітинці сітки. Наприклад, басейн на зображенні нижче відповідає зеленому прямокутнику, тоді як будівля відповідає фіолетовому.

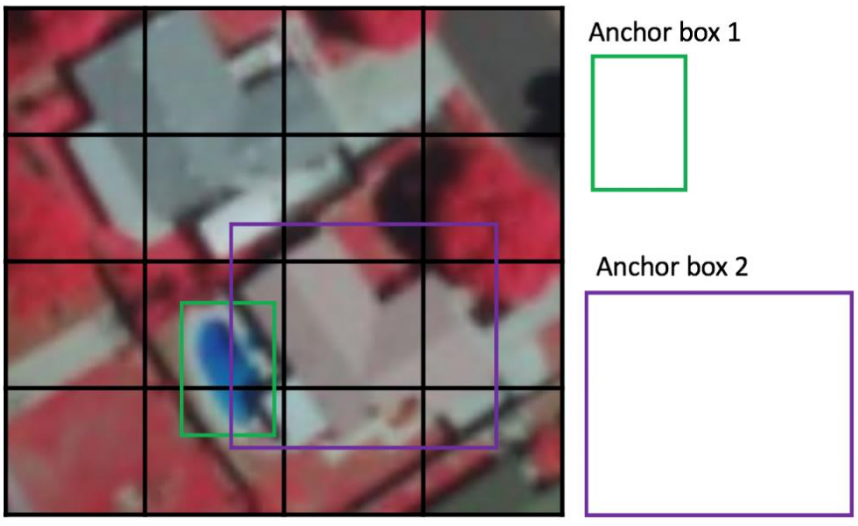


Рисунок 1.10 – Приклад прив’язок [9]

1.6 Аналіз протоколів для відеозв’язку

VoIP протокол використовувує IP протокол для передачі голосу та мультимедіа через інтернет мережу (глобальну Internet, локальну LAN). Для цього голос або картинка оцифровується, стискається і конвертується в IP пакети [9].

Проблеми, що виникають при реалізації протокола:

1. Якість голосу або відео - олскільки IP був розроблений для передачі даних, він не надає гарантій у реальному часі, а лише забезпечує найкраще обслуговування. Щоб голосовий зв’язок через IP став прийнятним для користувачів, затримка має бути меншою за порогове значення, і IETF (Internet Engineering Task Force) працює над цим аспектом. Щоб забезпечити хорошу якість голосу, ми можемо використовувати або ехоподавлення, пріоритетність пакетів (надання вищого пріоритету голосовим пакетам) або пряме виправлення помилок.
2. Сумісність - у загальнодоступному мережевому середовищі продукти від різних постачальників повинні працювати один з одним, щоб передача голосу через IP стала загальною для користувачів. Для досягнення сумісності розробляються стандарти, і найпоширенішим стандартом для VOIP є стандарт H.323
3. Безпека – ця проблема існує, оскільки в Інтернеті будь-хто може перехопити пакети, призначені для когось іншого. Деяку безпеку можна забезпечити за допомогою шифрування та тунелювання. Загальний протокол тунелювання, який використовується, — це протокол Layer 2 Tunneling, а загальний механізм шифрування — це Secure Sockets Layer (SSL)
4. Масштабованість - оскільки дослідники працюють над тим, щоб забезпечити таку ж якість через IP, як і звичайні телефонні дзвінки, але за значно нижчою ціною, існує великий потенціал для високих темпів зростання систем VOIP. Системи VOIP мають бути достатньо гнучкими,

щоб вирости на ринок великих користувачів і дозволяти поєднання приватних і державних послуг.

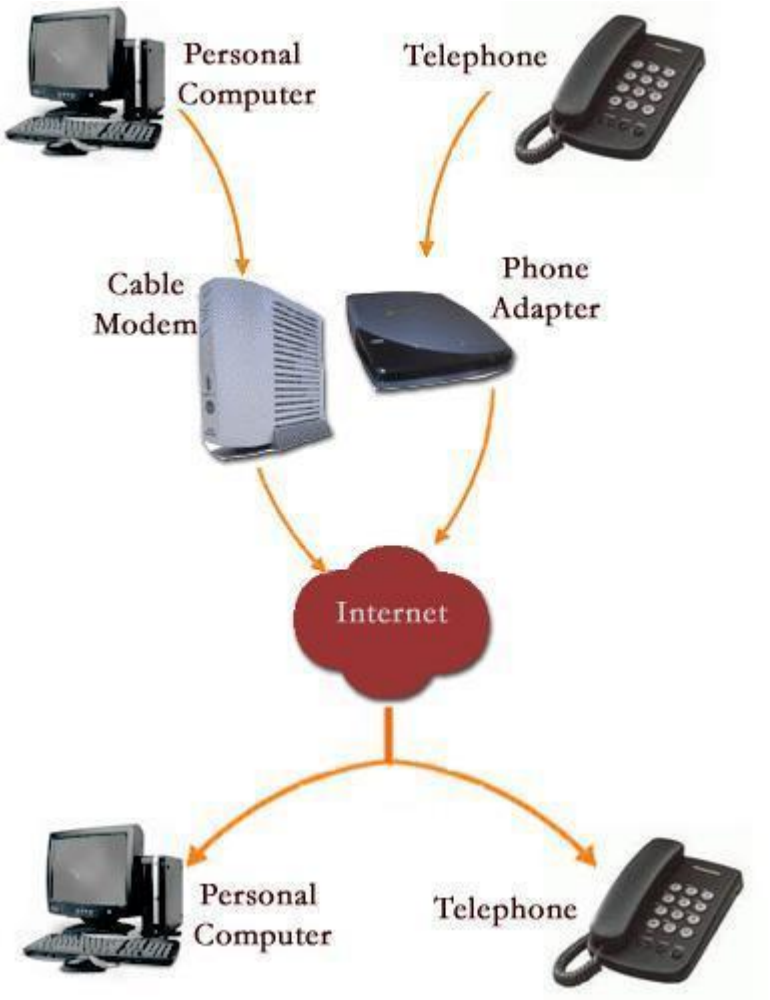


Рисунок 1.11 – Схема мережі девайсів, яка реалізує протокол VoIP [10]

H.323 STANDARD - Це стандарт ITU-T (Міжнародний союз телекомунікацій), якого повинні дотримуватися постачальники, надаючи послугу передачі голосу через IP [9]. Ця рекомендація містить технічні вимоги для голосового зв’язку через локальні мережі, припускаючи, що локальні мережі не забезпечують якість обслуговування (QoS). Спочатку він був розроблений для мультимедійних конференцій у локальних мережах, але пізніше був розширений, щоб охопити голос через IP. Перша версія була випущена в 1996 році, а друга версія H.323 набула чинності в січні 1998 року. Стандарт охоплює як зв'язок «точка-точка», так і багатоточкові конференції. Продукти та програми різних постачальників можуть взаємодіяти, якщо вони відповідають специфікації H.323.

Компоненти H.323:

1. Термінали – LAN кінцеві точки клієнтів, які забезпечують в режимі реального часу комунікацію
2. Гейтвеї - це кінцева точка в мережі, яка забезпечує двосторонній зв'язок у режимі реального часу між терміналами H.323 у IP-мережі та іншими терміналами ITU в комутованій мережі або до іншого шлюзу H.323. Вони виконують функцію «транслятора», тобто виконують переклад між різними форматами передачі, наприклад, з H.225 на H.221. Вони також здатні здійснювати переклад між аудіо- та відеокодеками.
3. Гейткіпери - це найважливіший компонент системи H.323, який виконує обов'язки «менеджера». Він діє як центральна точка для всіх викликів у межах своєї зони (зона — це агрегація гейткіпера та зареєстрованих у ньому кінцевих точок) і надає послуги зареєстрованим кінцевим точкам.
4. MCU - це кінцева точка в мережі, яка забезпечує можливість трьох або більше терміналів і шлюзів брати участь у багатоточковій конференції. MCU складається з обов'язкового багатоточкового контролера (MC) і додаткових багатоточкових процесорів (MP). MC визначає загальні можливості терміналів за допомогою H.245, але не виконує мультиплексування аудіо, відео та даних. Мультиплексування медіа-потоків виконується MP під керуванням MC.

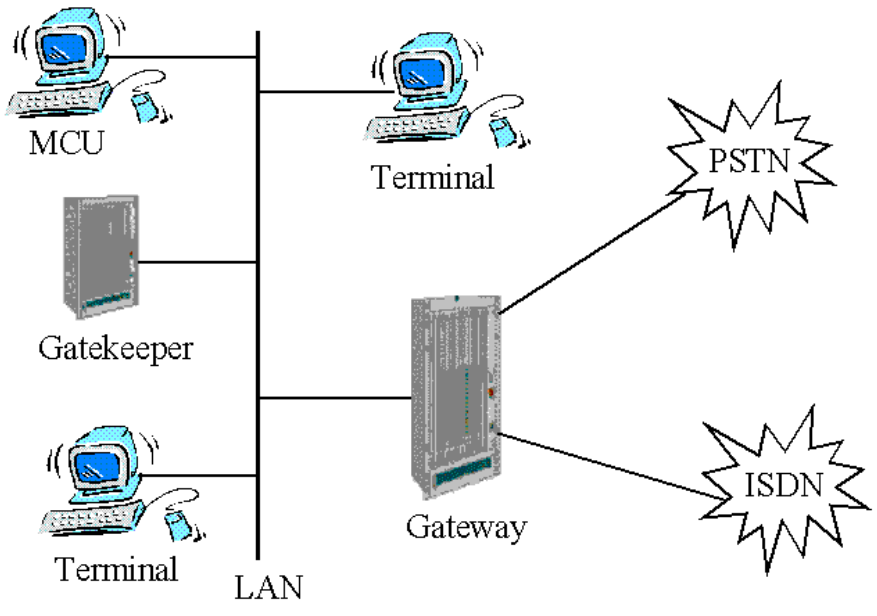


Рисунок 1.12 – Компоненти H.323 [11]

Аудіо-, відео- та реєстраційні пакети використовують ненадійний протокол дейтаграм користувача (UDP), тоді як пакети даних і програм керування використовують надійний протокол керування передачею (TCP) як транспортний протокол. Крім протоколу T.120, інші протоколи описані в статті. Протокол T.120 використовується для визначення частини даних конференц-зв'язку [11].

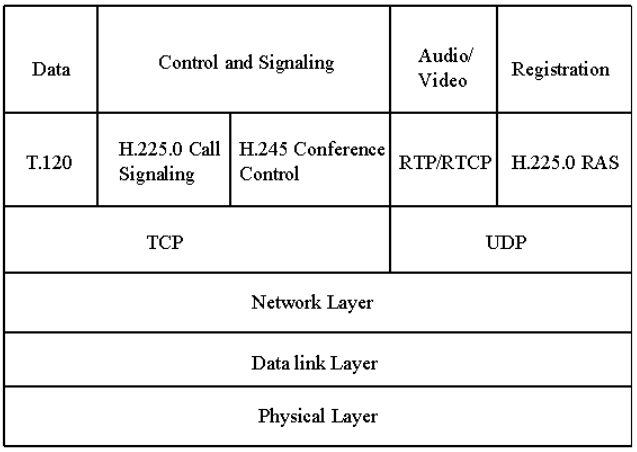


Рисунок 1.14 – Стек H.323 [11]

Канал сигналізації виклику використовується для передачі керуючих повідомлень H.225 [11]. У мережах, які не містять шлюз, повідомлення сигналізації про виклик передаються безпосередньо між абонентом і викликаною кінцевою точкою за допомогою транспортних адрес сигналізації виклику. Передбачається, що кінцева точка, що викликає, знає транспортну адресу сигналізації виклику кінцевої точки, що викликається, і тому може спілкуватися безпосередньо. У мережах, які містять гейткіпер, початковий обмін повідомленнями про доступ відбувається між викликаючою кінцевою точкою та гейткіпером за допомогою транспортної адреси каналу RAS гейткіпера. Сигналізація виклику здійснюється через TCP (надійний канал).

Повідомлення сигналізації про виклик можна передати двома способами. Перший спосіб – це маршрутизація сигналів про виклик Gatekeeper Routed Call Signaling, коли сигнальні повідомлення про виклик маршрутизуються через

Gatekeeper між кінцевими точками. Іншою альтернативою є пряма сигналізація виклику кінцевої точки, коли повідомлення сигналізації про виклик передаються безпосередньо між кінцевими точками. Повідомлення про доступ обмінюються з гейткіпером по каналу RAS, після чого відбувається обмін повідомленнями сигналізації виклику по каналу сигналізації виклику, за яким, у свою чергу, слідує встановлення каналу керування H.245.

Контроль маршрутизації каналу

Коли використовується сигналізація виклику, що направляється гейткіпером, існує два методи маршрутизації каналу керування H.245. Перший варіант встановлює канал керування H.245 безпосередньо між кінцевими точками, тоді як у другому випадку встановлення каналу керування H.245 здійснюється через гейткіпер.

Існують і інші різновиди протоколу VoIP:

1. SESSION INITIATION PROTOCOL (SIP)
2. Media Gateway Control Protocol(MGCP)
3. RTP and RTCP (Real-time Transport Protocol and Real-time Control Protocol)

1.7 Протокол WebRTC

WebRTC - це стандарт, який забезпечує одноранговий зв’язок у режимі реального часу та обмін медіа-даними в браузерах, усуваючи потребу завантажувати та встановлювати додаткові програми чи доповнення [13]. Завдяки WebRTC будь-який браузер може працювати як кінцева точка відеоконференції: вам потрібно лише відкрити веб-сторінку зустрічі, щоб почати відеоконференцію. У цій статті описано WebRTC, популярні випадки його використання, а також переваги та недоліки цієї технології.

Суть роботи протоколу:

1. Юзер відкриває сторінку відеодзвінка/відеоконференції

2. Браузер може запитувати доступ до веб-камери та мікрофона. У цьому випадку користувач повинен надати дозвіл програмі WebRTC на доступ до пристроїв користувача. Бувають також випадки, коли цей дозвіл не потрібен, напр. під час перегляду прямої трансляції.
3. Session Description Protocol (SDP) пакет генерується в браузері, який ініціює з’єднання. Насправді це текстовий файл, який містить важливі деталі підключення, напр. тип медіаданих (аудіо, відео чи контент), кодеки, які параметри підтримує браузер тощо.
4. Залежно від того, як реалізована технологія, ініціатор з'єднання передає цей пакет іншим учасникам. Для цього часто використовується сигнальний сервер і протокол WebSocket.
5. На стороні приймача браузер отримує SDP-пакет, а потім генерує подібний, який також приймає дані з першого пакету. Другий пакет надсилається назад на ініціальну сторону. Зараз обидва клієнти вже в чомусь пізнали один одного.
6. Залежно від його реалізації разом із попередніми кроками аналізується стан підключення до мережі. Клієнтам надається адреса сервера STUN, яка використовується для визначення зовнішньої IP-адреси пристрою. Потім він порівнюється з внутрішньою IP-адресою, щоб визначити, чи використовується NAT із з’єднанням і, якщо так, як маршрутизуються пакети UDP. У більш складних випадках, напр. коли використовується подвійний NAT, розробники використовують сервери TURN. По суті, це повторювачі, які перетворюють однорангове (P2P) з’єднання на клієнт-сервер-клієнт.
7. Після успішного виконання цих кроків підключення буде встановлено. Подія onicecandidat періодично викликається для передачі інформації про IP-адреси, налаштування NAT і спроби підключення між клієнтами.

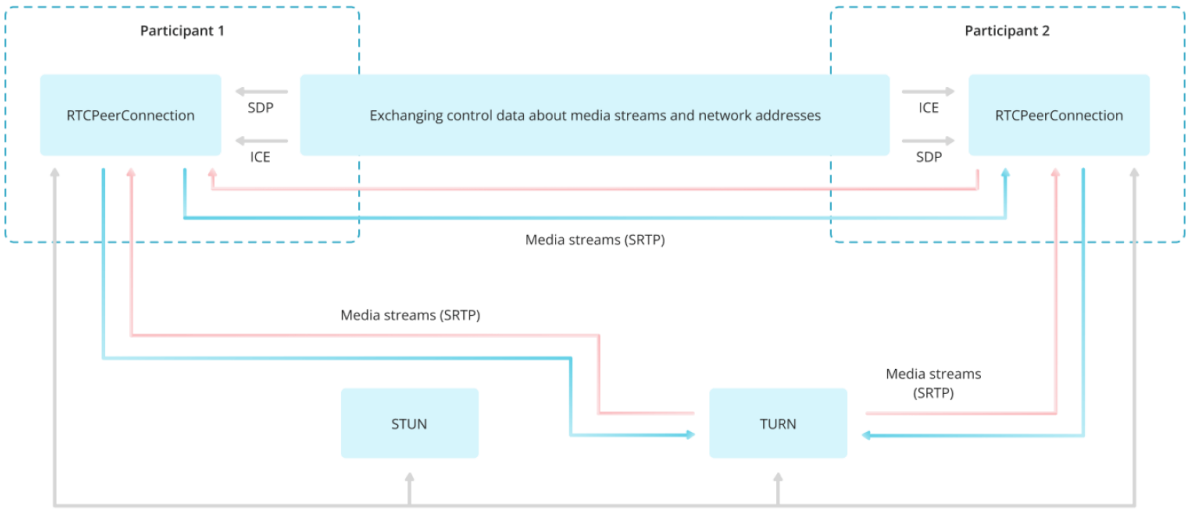


Рисунок 1.15 – Схема протокола WebRTC [14]

Переваги протоколу WebRTC

1. Не потребує інсталяції нових програм окрім браузера
2. Високоякісний рівень комунікації
3. Сильний акцент на безпеку: усі з'єднання захищені та зашифровані відповідно до протоколів DTLS і SRTP. При цьому WebRTC працює тільки за протоколом HTTPS, а сайт, що використовує цю технологію, повинен мати підписаний сертифікат.
4. Інтерфейси керування на основі HTML5 і JavaScript.
5. Кросплатформенність: програма WebRTC однаково добре працює з будь-якою операційною системою — настільною чи мобільною — якщо браузер підтримує WebRTC. Це значно економить ресурси розробки програмного забезпечення.

Недоліки

1. Рішення WebRTC несумісні одне з одним, оскільки стандарт описує лише методи передачі відео- та аудіоданих, дозволяючи розробникам вирішувати методи адресації, параметри відстеження стану, обмін повідомленнями та файлами, планування тощо.

2. WebRTC визначає справжні IP адреси користувачів, що є фактором небезпеки.
3. WebRTC не підтримує віддалене керування робочим столом.

WebRTC використовують у таких популярних веб додатках як GoogleMeet, Jitsi Meet та BigBlueButton.

Отже, підсумовуючи, можна сказати, що для вирішення таких вимог як встановлення комунікації здорової та глухонімої людини, можна побудувати модель нейромережі на базі алгоритмів *R-CNN і Fast(er), YOLO або SSD.* Нейромережа дозволить перевести зображення жестів у голос або текст. Систама відеозв'язку має бути заснована на протоколі WebRTC. Цей протокол відповідає вимогам кросплатформеності та спроможний захистити дані користувача. Проаналізувавши існуючі додатки-месенджери, можна з впевнено сказати, що додаток має мати зручний і простий інтерфейс, для створення викликів та листування.

РОЗДІЛ 2 ПРОЕКТНІ РІШЕННЯ. АРХІТЕКТУРА ДОДАТКА ТА ІНТЕГРАЦІЯ ПОТРІБНИХ ТЕХНОЛОГІЙ

2.1 Короткий опис дій, які були провдені для досягнення мети роботи

Для досягнення мети роботи та відповідно до поставлених функціональних та нефункціональних вимог, було проведено ретельний аналіз предметної області та наявних рішень. Особлива увага приділялась розробці SSD моделі штучного інтелекту та інтеграції з нею засобів зв'язку на базі WebRTC.

2.2 Навчання SSD моделі

Для побудови SSD моделі для розпізнавання образів було вирішено використовувати open-source бібліотеку Tensorflow для мови програмування Python [13]. Ця бібліотека дозволяє швидко та зручно побудувати модель для виконання завдання розпізнавання образів. Для цього необхідно зібрати тренувальний та перевірочний набори зображень, які повинні бути у форматі Pascal VOC або COCO.

Для тренування моделі необхідно завчасно позначити області на зображеннях, які повинні бути розпізнані, що називається "labelimg". Для цієї мети буде використовуватися програма LabelImg [14]. Її робота полягає у виділенні прямокутної області на зображенні, яку потрібно розпізнати та позначити її підписом.

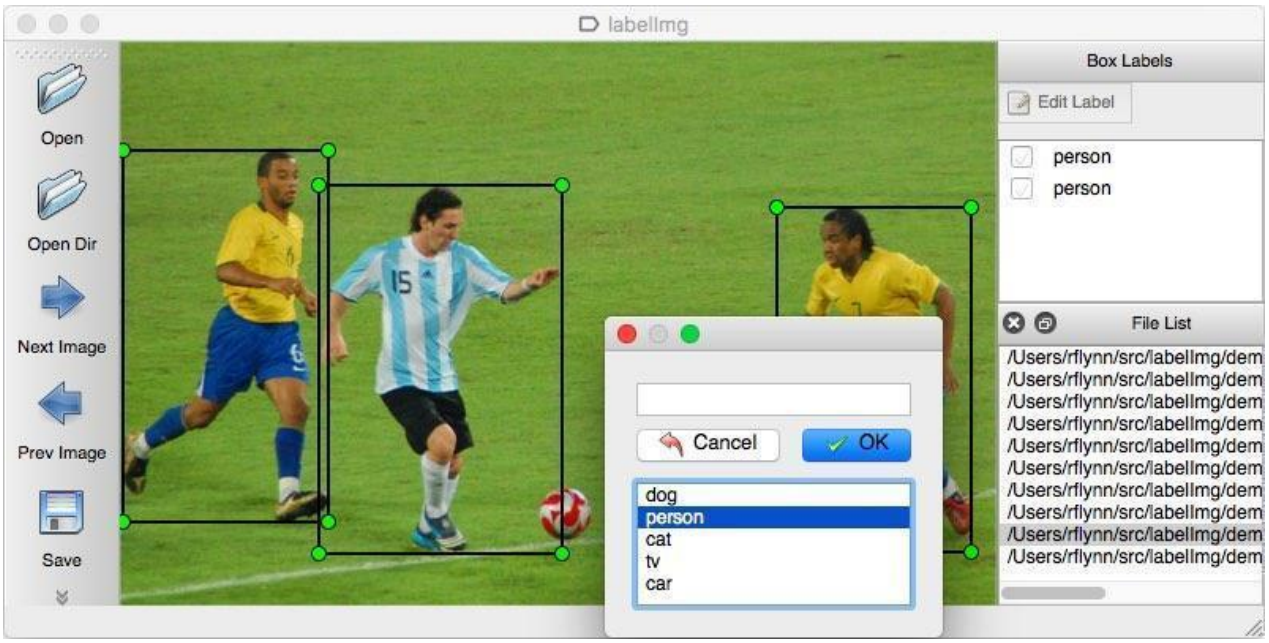


Рисунок 2.1 – Приклад позначення області в програмі LabelImg [16]

Для запуску навчання моделі також потрібно налаштувати конфігураційний файл, в якому необхідно вказати кількість класів для розпізнавання. Цей параметр вказує кількість різних класів, які може

розпізнати модель. Вказується розмір зображення, яке буде використовуватись для навчання моделі. Кількість анкорів (anchors). Анкори використовуються для вибору областей зображення, які мають бути розпізнані. Визначається кількість шарів та їхні параметри. Вказується швидкість навчання, кількість ітерацій тощо. Шлях до тренувальних та перевірочних наборів даних.

2.3 Інтеграція WebRTC

WebRTC інтегрується у веб-додаток за допомогою JavaScript. Наразі більшість сучасних браузерів підтримають цей протокол.

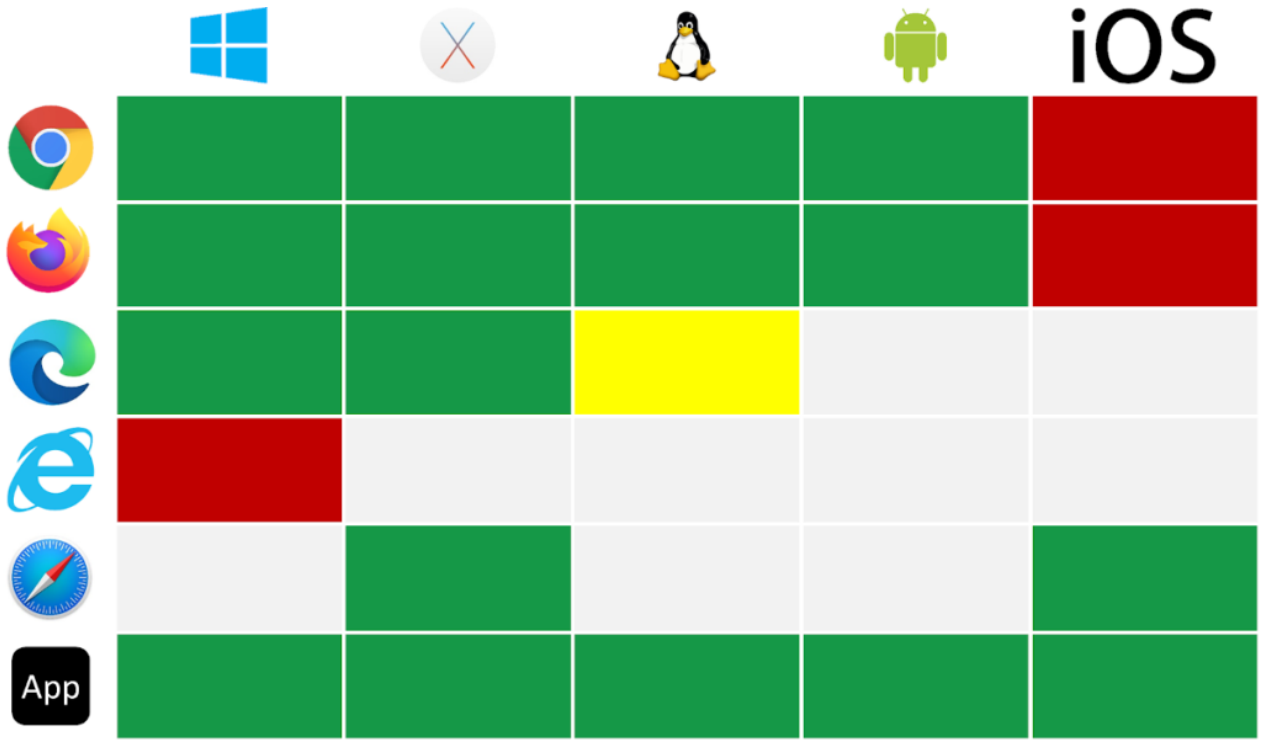


Рисунок 2.2 – Таблиця, що показує підтримувані платформи та браузери для WebRTC [17]

Додатки на основі WebRTC повинні виконувати кілька завдань:

1. Отримувати потік аудіо, відео та інших даних.

2. Отримувати мережеву інформацію, таку як IP-адреси та порти, та обмінюватися нею з іншими веб-клієнтами WebRTC (відомими як піри) для забезпечення підключення, навіть через NAT та брандмауери.
3. Координувати сигнальну комунікацію для повідомлення про помилки та ініціювання або закриття сесій.
4. Обмінюватися інформацією про медіа та можливості клієнта, такі як роздільна здатність та кодеки.
5. Комунікувати потоковий аудіо, відео або дані.

Для отримання та передачі потокових даних, WebRTC реалізує наступні API [13]:

MediaStream - отримання доступу до потоків даних, таких як з камери та мікрофону користувача. MediaStream API представляє синхронізовані потоки медіа. Наприклад, потік, взятий з камери та мікрофона, містить синхронізовані відео- та аудіо-доріжки. Повертається масив MediaStreamTracks за допомогою методів getAudioTracks() та getVideoTracks().

RTCPeerConnection - аудіо або відео-дзвінки з можливістю шифрування та керування пропускною здатністю.

RTCDataChannel - передача p2p загального типу даних.

WebRTC використовує RTCPeerConnection для обміну потоковими даними між браузерами (так званими пірами), але також потребує механізму для координації зв'язку та відправки контрольних повідомлень, процес, відомий як сигналізація. Методи та протоколи сигналізації не вказуються у WebRTC, сигналізація не є частиною RTCPeerConnection API.

В цій роботі сигналізація буде відбуватися за допомогою Firebase.

WebRTC клієнти (відомі як піри або А, Б) також потребують отримати та обмінятися локальною та віддаленою інформацією про аудіо та відео, таку як роздільна здатність та можливості кодеків. Сигналізація для обміну конфігурацією медіа виконується шляхом обміну пропозиції та відповіді з використанням протоколу опису сесій (SDP):

1. Клієнт А запускає метод RTCPeerConnection createOffer (). При цьому викликається зворотній виклик з аргументом RTCSessionDescription: локальний опис сесії клієнта А.

2. У зворотньому виклику кдієнт А встановлює локальний опис за допомогою setLocalDescription () та відправляє його опис сесії до клієнта Б через їх канал сигналізації (Firebase). RTCPeerConnection не почне збирати кандидатів, поки не буде викликано setLocalDescription ().
3. Клієнт Б встановлює опис, який клієнт А відправила йому, як віддалений опис за допомогою setRemoteDescription ().
4. Клієнт Б запускає метод RTCPeerConnection createAnswer (), передаючи йому віддалений опис, який він отримав від клієнта А, щоб згенерувати локальний опис, який сумісний з описом клієнта А. Зворотний виклик createAnswer () передається RTCSessionDescription: Клієнт Б встановлює його як локальний опис та відправляє його до клієнта А.
5. Коли клієнт А отримує опис сесії Б, він встановлює його як віддалений опис за допомогою setRemoteDescription.

   Таким чином клієнти тепер знають один про одного.

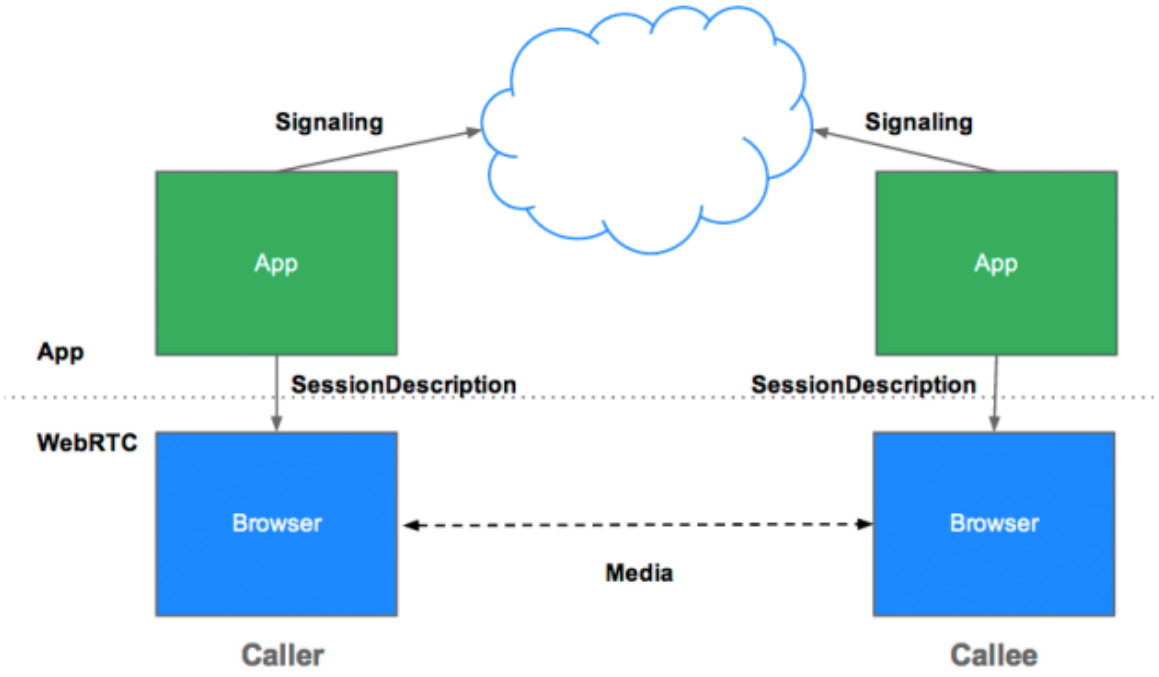


Рисунок 2.3 – Графічне представлення обміну SDP пакетами [18]

У реальному світі WebRTC потребує серверів, бо може статися

наступне:

1. Користувачі знаходять один одного і обмінюються "реальними" даними, такими як імена.
2. Клієнтські програми WebRTC (однорангові) обмінюються мережевою інформацією.
3. Однорангові програми обмінюються даними про медіа, такими як формат і роздільна здатність відео.
4. Клієнтські програми WebRTC проходять через NAT-шлюзи та брандмауери.

Іншими словами, WebRTC потребує чотирьох типів функціональності на стороні сервера:

1. Виявлення користувачів та комунікація.
2. Сигналізація.
3. Обхід NAT/брандмауерів.
4. Сервери-ретранслятори на випадок збою зв'язку між одноранговими серверами.

Щоб обходити NAT можно інтегрувати STUN сервер. Достатньо сказати, що протокол STUN і його розширення TURN використовуються фреймворком ICE для того, щоб дозволити RTCPeerConnection впоратися з обходом NAT та іншими проблемами.

ICE - це фреймворк для з'єднання однорангових пристроїв, наприклад, двох клієнтів відеочату. Спочатку ICE намагається з'єднати однорангові комп'ютери напряму, з найменшою можливою затримкою, через UDP. Під час цього процесу В цьому процесі STUN-сервери мають єдине завдання - дозволити одноранговому комп'ютеру, що знаходиться за NAT, дізнатися дізнатися його публічну адресу і порт.

Якщо UDP не працює, ICE пробує TCP: спочатку HTTP, потім HTTPS. Якщо пряме з'єднання не вдається – зокрема , через обхід корпоративного NAT і брандмауерів - ICE використовує проміжний (ретрансляційний) сервер TURN. Іншими словами, ICE спочатку використовує STUN з UDP для прямого з'єднання з одноранговими вузлами, а якщо це не вдасться, повернеться до ретрансляційного TURN серверу TURN. Вираз "пошук кандидатів" відноситься

до процесу пошуку мережевих інтерфейсів та портів.

Для даної роботи вирішено використовувати STUN сервер від Google.

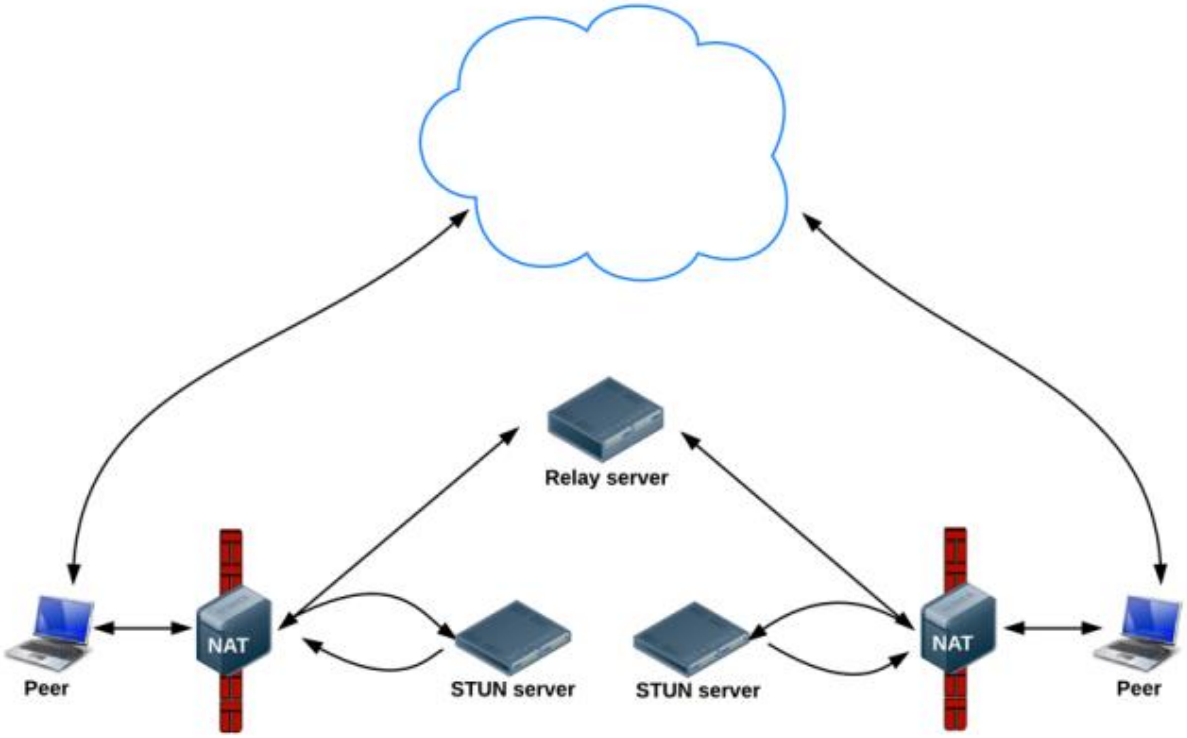


Рисунок 2.4 – Графічне представлення процесу пошука кандидатів [18]

2.4 Безпека WebRTC

Існує кілька способів, якими програма або плагін для спілкування в реальному часі може поставити під загрозу безпеку. Наприклад:

1. Незашифровані медіа або дані можуть бути перехоплені на шляху між браузерами, або між браузером і сервером.
2. Додаток може записувати і поширювати відео або аудіо без відома користувача.
3. Шкідливе програмне забезпечення або віруси можуть бути встановлені разом з нешкідливим на перший погляд плагіном або додатком.

WebRTC має декілька можливостей для уникнення цих проблем:

1. Реалізації WebRTC використовують безпечні протоколи, такі як DTLS та SRTP.
2. Шифрування є обов'язковим для всіх компонентів WebRTC, включаючи механізмів сигналізації.
3. WebRTC не є плагіном: його компоненти працюють з «під коробки браузера», а не в окремому процесі, компоненти не потребують окремого встановлення та оновлюються разом з оновленням браузера.
4. Доступ до камери та мікрофону повинен бути наданий явно, це чітко відображається користувачу.

2.5 Обробка потоку зображень для розпізнавання жестів

Щоб оброблювати жестові зображення та перекладати їх в режимі реального часу, було розглянуто два архітектурні підходи:

1. Оброблювати відеопотік через окремий проміжковий (проксі) сервер між клієнтом А та Б.
2. Оброблювати зображення на клієнтській стороні без додаткових серверів.

Було вирішено застосувати другий підхід, оскільки він доволі простий для імплементації та не вимагає значних додаткових ресурсів. Хоча перший підхід виглядає більш перспективним з точки зору централізованості та безпеки, оскільки з'єднання P2P, достатньо факту, що клієнт А довіряє клієнту Б, тому робити переклад зображення можна на кінцевому клієнті.

Отже виникає питання як класифікувати зображення з вашої веб-камери без сервера?

У березні 2018 року Google представив Tensorflow.js - бібліотеку з відкритим вихідним кодом, яку можна використовувати для визначення, навчання та запуску моделей машинного навчання повністю в браузері за допомогою Javascript.

Tensorflow.js має опцію захоплення вихідних зображень з різних html-

елементів, включаючи відеоелементи. Використовуючи цю опцію, ми можемо класифікувати зображення, що надходять з відеопотоків WebRTC.

Наприклад маємо відео елемент html сторінки:

<video id="gum-local" width="128" height="128" autoplay playsinline></video>

У заголовку імпортуємо бібліотеку

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@0.12.0"> </script>

Для того, щоб завантажити модель з каталогу jsmodel, виконуємо наступні дії:

```
tf.loadModel('jsmodel/model.json')
   .then(function (model) {

   })
```

Тепер потрібно взяти відеокадри з відеоелемента і додати їх до нашої моделі. В ідеалі це потрібно зробити для кожного відеокадру, але на практиці це сильно залежить від продуктивності комп'ютера/телефону. Тому викликаємо window.requestAnimationFrame і передамо наступну функцію

```
function onFrame(model, webrtcElement) {
  // Build a tensor from the frame image
  const tensor = tf.fromPixels(webrtcElement);
  // MobileNet model expects RGB values in 0..1 range for each component,
  // but the tensor contains 0..255 values for each RGB component.
  // So we must divide them by 256:
  const eTensor = tensor.expandDims(0).asType('float32').div(256.0);

  // Do actual prediction
  const pred = model.predict(eTensor);

  // Prediction returns an array with probablity which corresponds
  // to each class. We need to find the class with highest probability:
  max = tf.argMax(pred, 1)
```

```
  const index = max.get([0])

  // Now we should find which text corresponds to the found index
  // and that would be our classificaiton for the image.
}
```

В результаті отримуємо індексну змінну, значення якої відповідає передбаченому класу зображення.

2.6 Обробка голосу та його перетворення у текст

Перетворення голосу у текст дуже важливе для людини з вадами слуху, адже вона може вміти читати. В рамках цієї кваліфікаційної роботи вирішено було скористатися сервісом від Google Cloud, а саме «Speech-to-Text» [19]. Сервіс було обрано через його надійність та ключові переваги, такі як:

1. Адаптація мовлення - можливість надання підказок для підвищення точності транскрипції рідкісних і специфічних слів або фраз. Використання класів для автоматичного перетворення вимовлених чисел на адреси, роки, валюти тощо.
2. Моделі для конкретної галузі - можливість обирати з низки підготовлених моделей для голосового керування, телефонних дзвінків і транскрипції відео, оптимізованих під вимоги до якості в конкретній галузі.
3. Легко порівнювати якість - можливість експериментувати зі своїм мовним звуком за допомогою простого у використанні інтерфейсу.
4. Мовлення на пристрої – можливість запускати мовні алгоритми Google Cloud локально на будь-якому пристрої, незалежно від підключення до Інтернету.
5. Підтримка понад 100 мов та діалектів.

Перетворення мови в текст має три основні методи розпізнавання мовлення. Вони наведені нижче:

1. Синхронне розпізнавання (REST і gRPC) надсилає аудіодані до Speech-to-Text API, виконує розпізнавання цих даних і повертає результати після обробки всього аудіо. Запити на синхронне розпізнавання обмежуються аудіоданими тривалістю не більше 1 хвилини.
2. Асинхронне розпізнавання (REST і gRPC) надсилає аудіодані до API перетворення мови в текст і ініціює довготривалу операцію. Використовуючи цю операцію, ви можете періодично опитувати результати розпізнавання. Підходить для асинхронних запитів для аудіоданих будь-якої тривалості до 480 хвилин.
3. Потокове розпізнавання (тільки gRPC) виконує розпізнавання аудіоданих, наданих у двонаправленому потоці gRPC. Потокові запити призначені для розпізнавання в реальному часі, наприклад, для захоплення звуку з мікрофона в реальному часі. Розпізнавання потокового мовлення надає проміжні результати під час захоплення аудіо, що дає змогу отримати результат, наприклад, поки користувач ще говорить.

   Для проекту вирішено використовувати потокове розпізнавання.

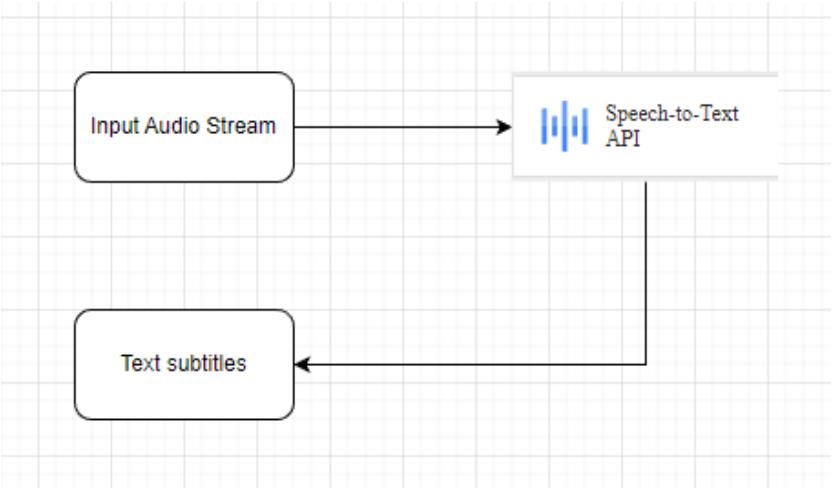


Рисунок 2.5 – Процес перетворення аудіо-потоку в субтитри

2.7 Проектування додатка

Першим кроком розробки проекту стала разробка архітектури. Відповідно до функціональних вимог було створено дерево функцій мобільного додатку:

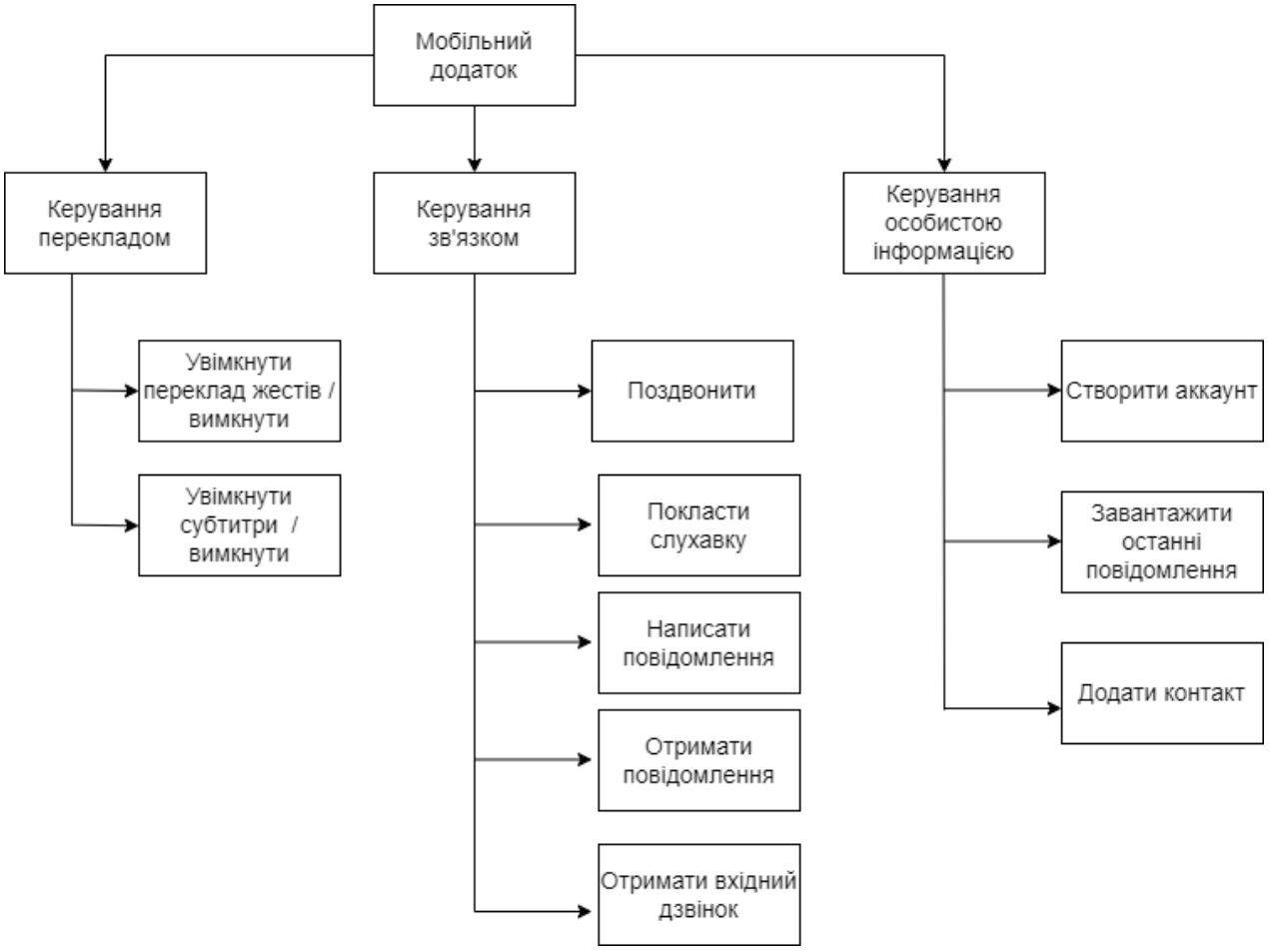


Рисунок 2.6 – Функціональний аналіз мобільного додатку

Оскільки для додатку необхідно створити модель нейромережі для розпзнавання образів, було вирішено створити декілька модулів (компонентів) системи.

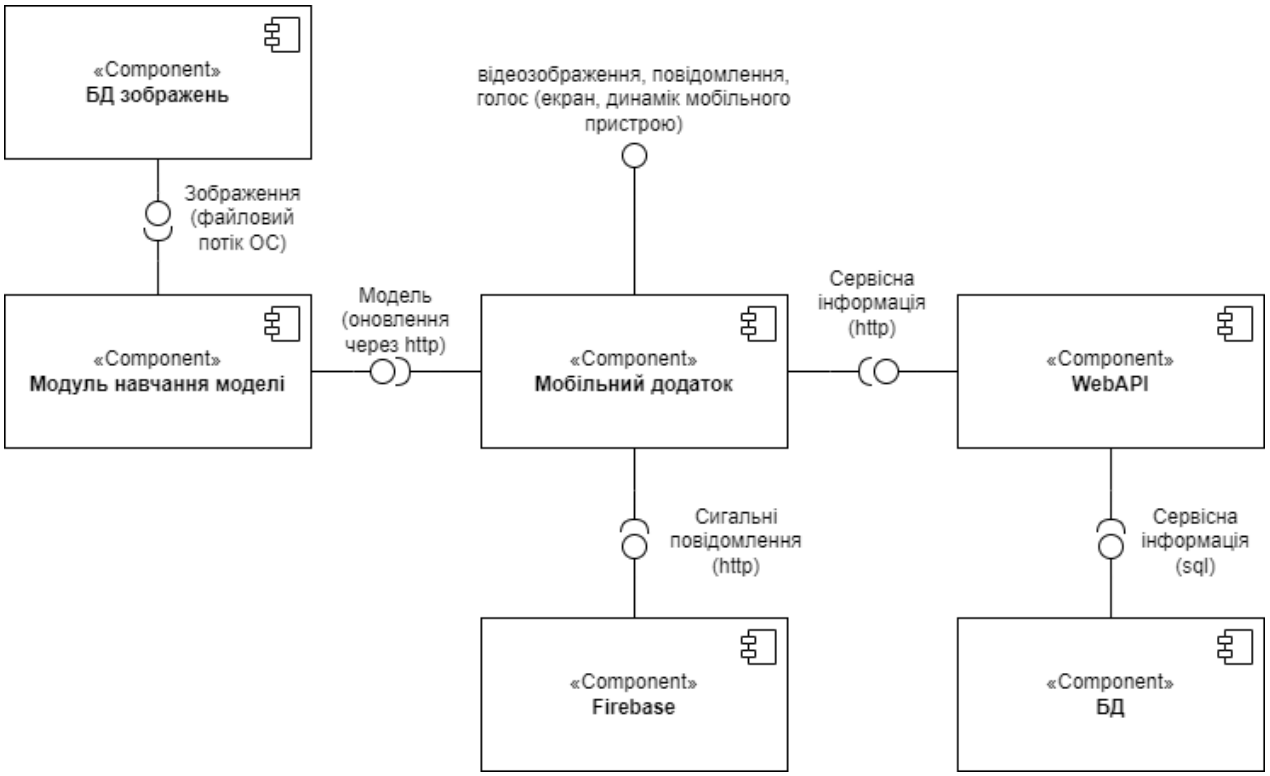


Рисунок 2.7 – Діаграма компонентів системи

Було виділено всього 6 основних компонентів системи:

1. Мобільний додаток - головний компонент системи, як вхідні дані потребує модель нейроної мережі, сервісну інформацію (дані користувача, сесії), сигналізацію (повідомлення про дзвінок). На виході - відеозображення, повідомлення, голос. Мова програмування JS.
2. WebAPI - проміжний компонент для організації обміну даними із БД через HTTP протокол. На вході сервісна інформація (дані користувача, сесії), на виході також сервісна інформація. Мова програмування C#.
3. БД - сховище даних користувача (Ім’я, прізвище, телефон, пошта, ідентифікатор) та поточних сесій (ідентифікатор, початок - кінець сесії). На виході ці дані.
4. Firebase - сервіс від Google, який на виході та на вході дає/приймає сигнали про вхідні/вихідні виклики користувачів.
5. Модуль начання моделі - консольний застосунок для навчання моделі нейромережі. На виході SSD модель, на вході зображення жестів Мова програмування JavaScript.
6. БД зображень - скоріш за все звичайний датасет зображень для навчання.

Алгоритм роботи всієї системи:

1. Клієнт А створює контракт SDP (Session Descript Protocol), який описує P2P підключення (час, кодек і т.п.)
2. Контракт зберігається в сігналізуючому сервері Firebase.
3. Id контракта використовується для підключення іншим клієнтом Б. Клієнт Б формує відповідь і зберігає її в Firebase.
4. Firebase сигналізує клієнту А про те, що клієнт Б надав відповідь. Формується з’єднання.
5. Також оскільки, пряме підключення (P2P) неможливе без обходу фаєрволів і NAT, додаток використовує стандарт ICE (Interactive Connectivity Establishment). Клієнт А і Б формує списки ICE кандидатів, які містять в собі ip - адреси та порти доступні для підключення. WebRTC протокол отримує такі списки в фоновому режимі звертаючись до STUN сервера (наприклад Google). Отримуючи ці списки, клієнти зберігають їх у Firebase, інші клієнти відслідковують зміни, і коли вони отримують нові списки кандидатів, то алгоритм WebRTC вибирає найкращого для з’єднання.
6. Отримавши зображення клієнт А оброблює за допомогою моделі розпізнавання образів та генерує текст (субтитри)
7. Отримавши голос, клієнт Б оброблює його за допомогою Google cloud API та генерує текст (субтитри).

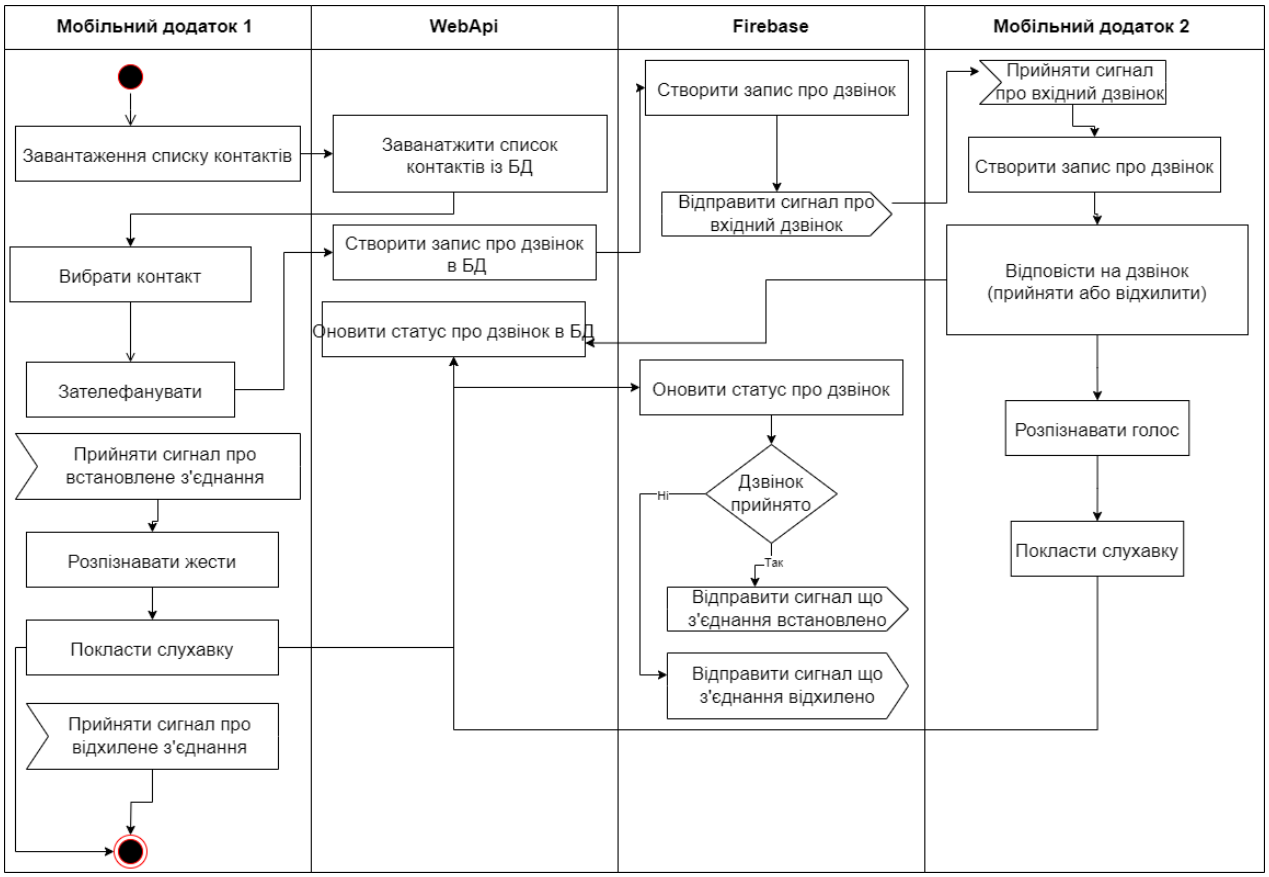


Рисунок 2.8 – Діагарама діяльності системи

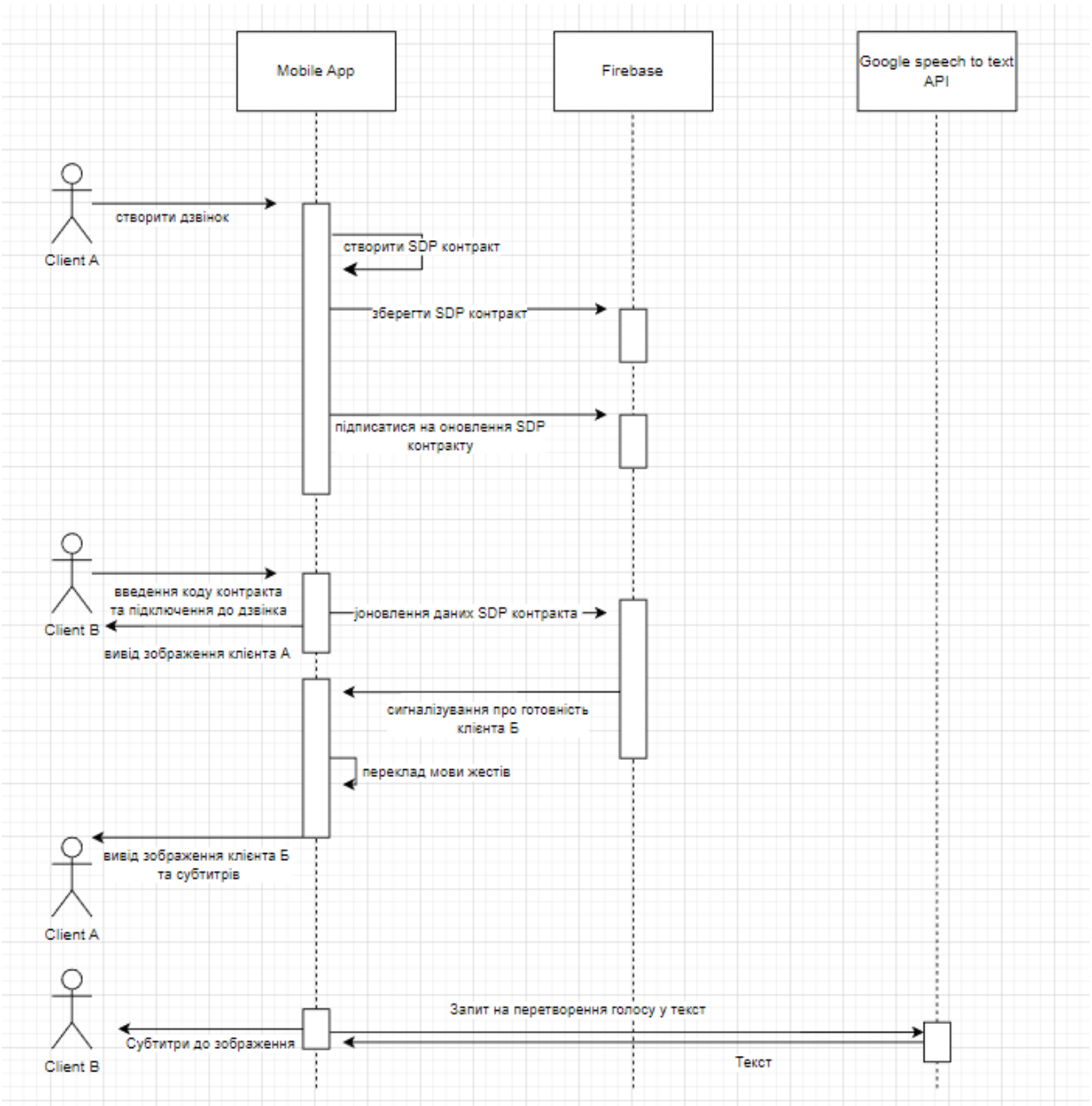


Рисунок 2.9 – Діаграма послідовності процесу з’єднання двох клієнтів та обробки їх відеопотоків

Для компонента “WebAPI” вирішено створити дві сутності такі як User (користувач), Call (дзвінок) та Message (Повідомлення).

User буде складатися з таких полів як

Id - тип guid - Id користувача

Firstname - тип string - ім’я користувача

Lastname - тип string - прізвище користувача

Nickname- тип string - нікнейм користувача

IsDeafmute- тип bool - чи користувач є глухонімою людиною

CreatedAt - тип DateTime - дата створення користувача

UpdatedAt - тип DateTime - дата онговлення інформації користувача

Call буде складатися з таких полів як

Id - тип guid - Id дзвінка

FromPersonId - тип guid - Id користуваща який телефанує

ToPersonId - тип guid - Id користуваща якому телефанують

Status - тип Status - Статус дзвінка (“Прийнято”, “Відхилено”, “Закінчився”)

CreatedAt - тип DateTime - дата створення дзвінка

UpdatedAt - тип DateTime - дата оновлення інформації дзвінка

Message буде складатися з таких полів як

Id - тип guid - Id повідомлення

FromPersonId - тип guid - Id користуваща який шле повідомлення

ToPersonId - тип guid - Id користуваща який приймає повідомлення

Text- Status - тип Текст повідомлення

Status - тип string - Статус повідомлення (“Доставлено”, “Прочитано”, “Не доставлено”)

CreatedAt - тип DateTime - дата створення дзвінка

UpdatedAt - тип DateTime - дата оновлення інформації дзвінка

Contact буде складатися з таких полів як

Id - тип guid - Id контакта

Firstname - тип string - ім’я користувача

Lastname - тип string - прізвище користувача

UserId- тип guid - Id користувача

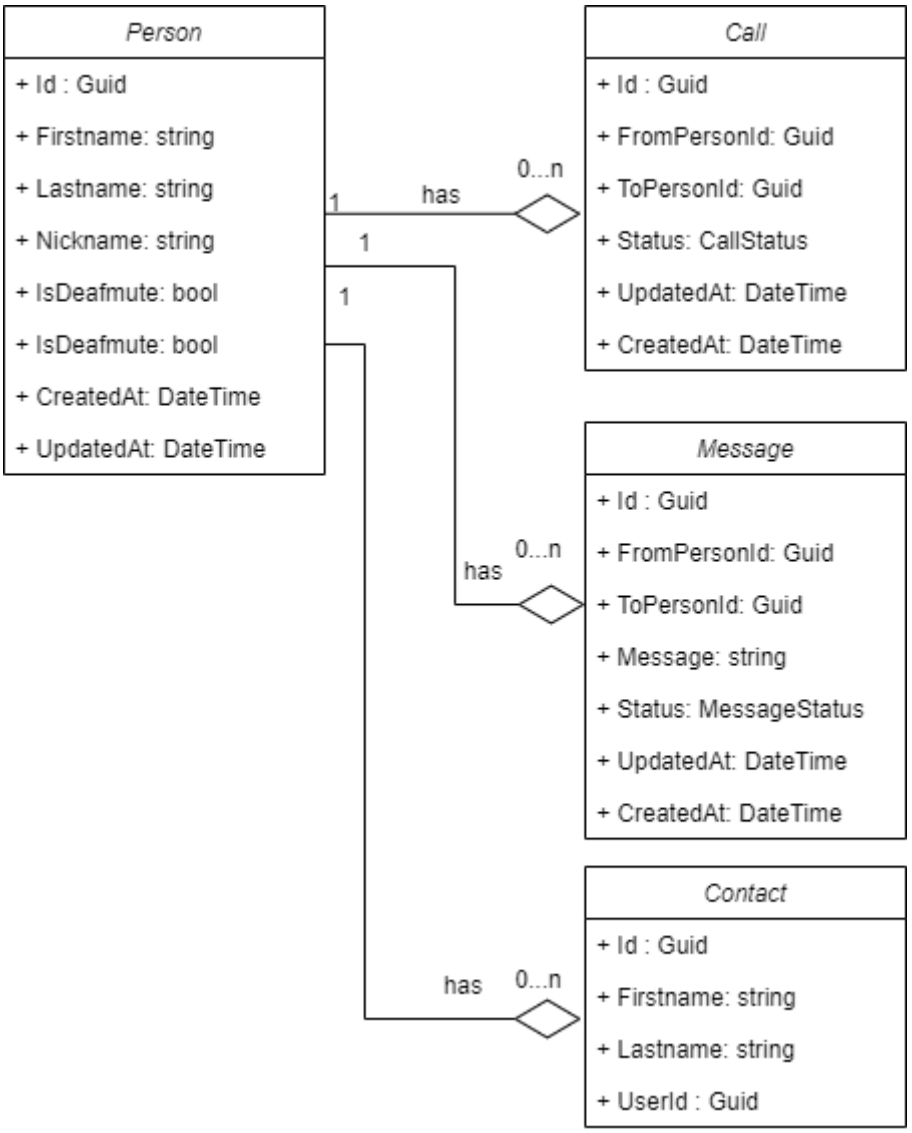


Рисунок 2.10 – UML концептуальна модель предметної області

Отже, в цьому розділі було визначено інструменти для навчання нейроної мережі такі як LabelImg та бібліотека TensorFlow. Визначено основні компоненти системи: мобільний додаток, модуль навчання нейроної мережі, бд зображень, Web API сервіс, БД та Firebase. Побудовано діаграму даних компонентів. Окреслено порти їх інтерфейсів (вхідні та вихідні дані). Побудовано дерево функцій системи, а також узагальнений алгоритм роботи системи. Побудовано діаграму класів сутностей необхідних для WebAPI.
Визначено стек основних технологій та мов програмування необхідних для компонентів системи:

1. Мобільний додаток - JS, React native, Tensorflow JS
2. Модуль навчання нейроної мережі - python, Tensorflow

3. Web API - dotNet, c#, Entity Framework
4. БД - Microsoft SQL server.

В результаті утворилась концепція системи для розпізнавання жестів мови глухонімих людини під час відеозв’язку.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ МЕСЕНДЖЕРА ТА МОДУЛЯ НАВЧАННЯ НЕЙРОМЕРЕЖІ РОЗПІЗНАВАННЯ ЖЕСТІВ

3.1 Реалізація мобільного додатку

В рамках цієї роботи було вирішено розробляти MVP (Minimum viable product) мобільного додатка, який буде відображати основні вимоги, які були затверджені на початку цієї роботи. Впершу чергу перед безпосередньо реалізацією мобільного додатка було створено концепти його інтерфейса. А саме: концепт екранів авторизації, екран контактів, екран виклику та екран відеозв’язку.

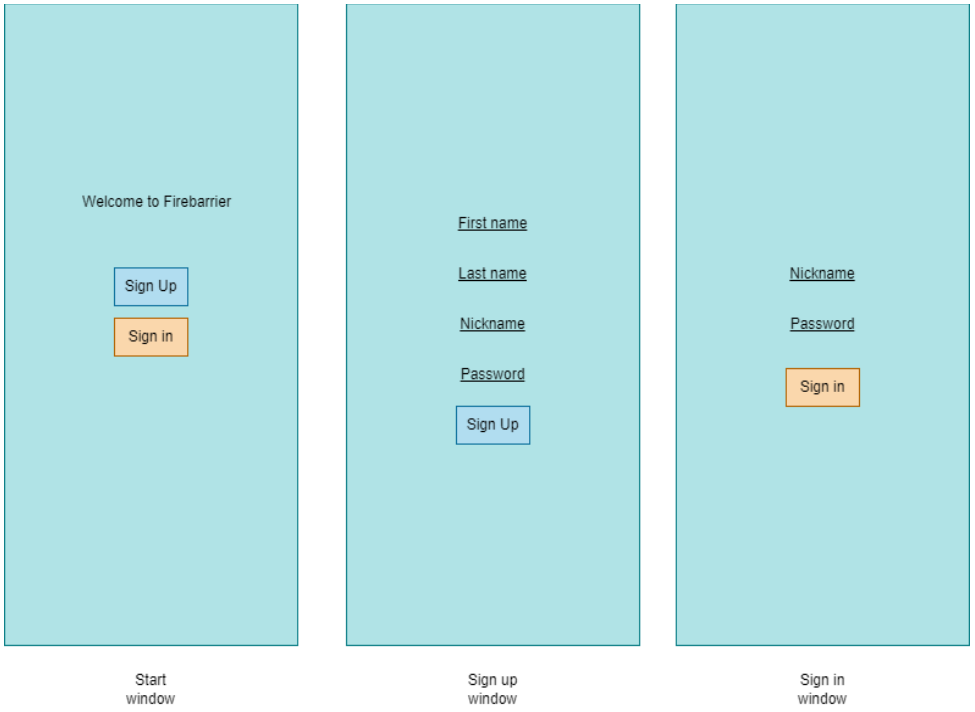


Рисунок 3.1 – Концепт екранів авторизації

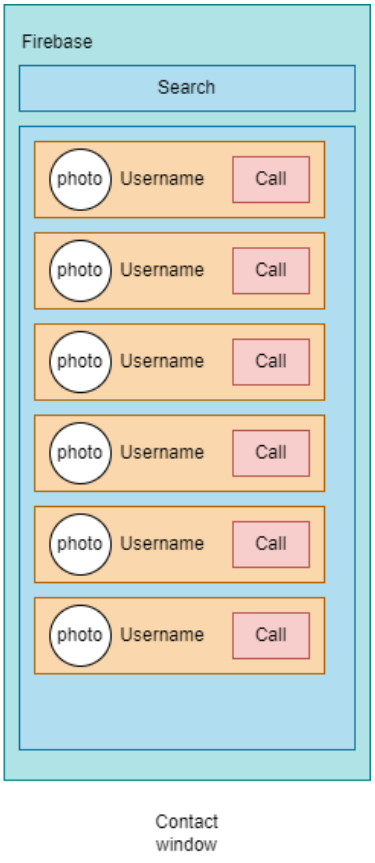


Рисунок 3.2 – Концепт екрану контактів

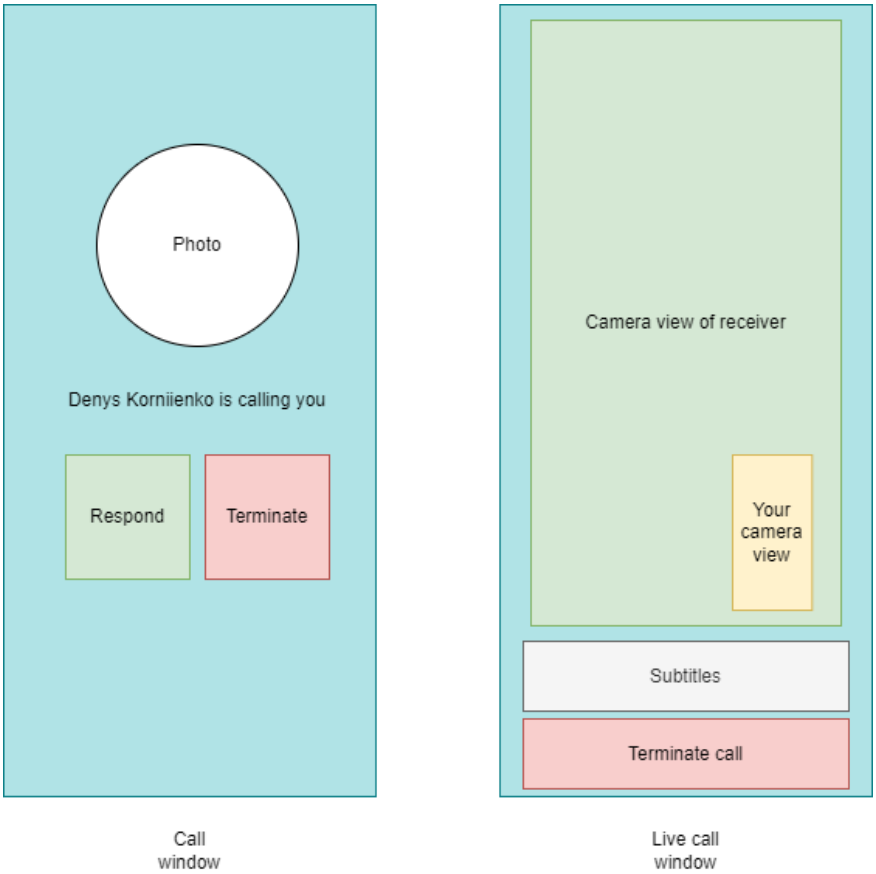


Рисунок 3.3 – Концепт екрану виклику та дзвінка

Для побудови мобільного додатку було завантажено і встановлено платформу Node.js версії 18.16.0 [20].

Далі для роботи з React Native було встановлено пакет expo командою

```
npm install -g expo-cli
```

Створив 3 компоненти react які відображають ці екрани

StartWindow.js

SignUpWindow.js

SignInWindow.js

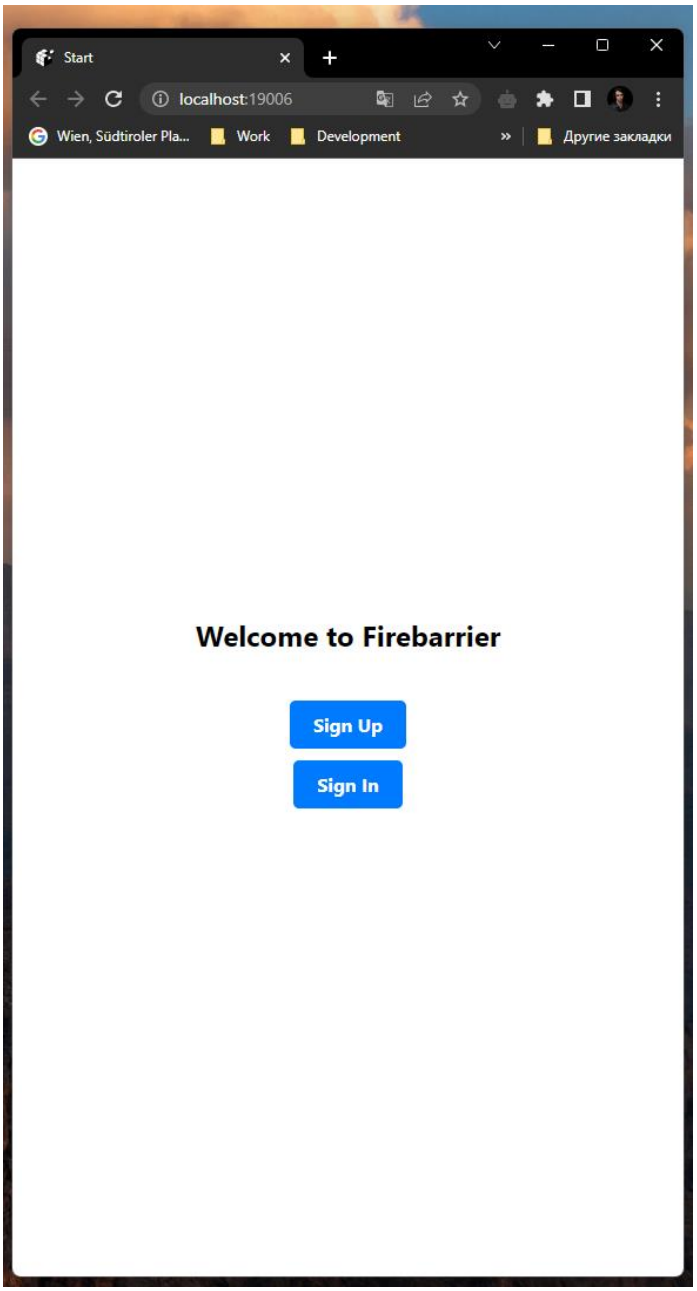


Рисунок 3.4 – Головний екран StartWindow

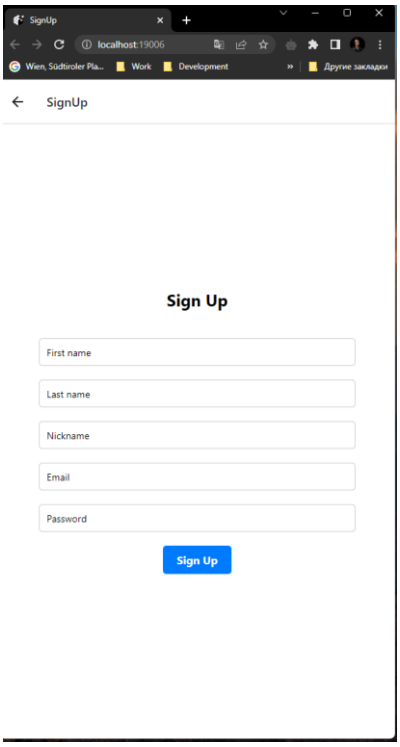


Рисунок 3.5 – Екран SignUpWindow

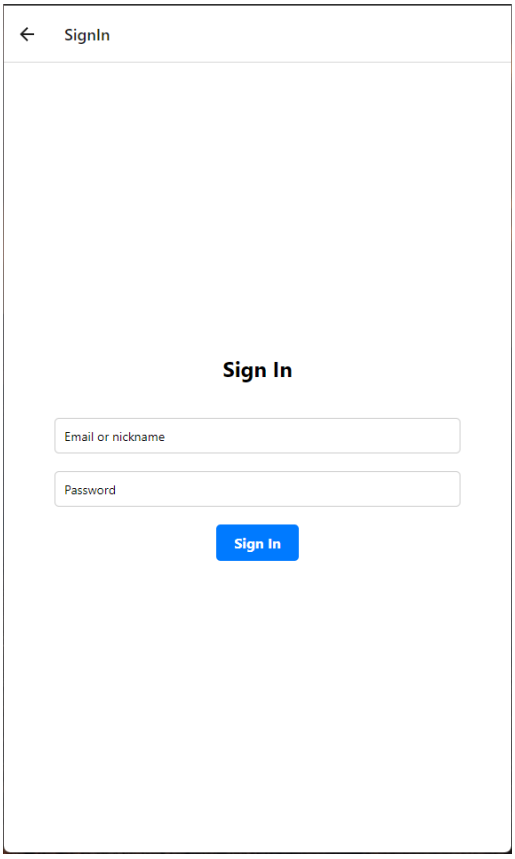


Рисунок 3.6 – Екран SignInWindow

Далі відповідно створив екрани контактів, вхідного дзвінка та дзвінка відповідно.

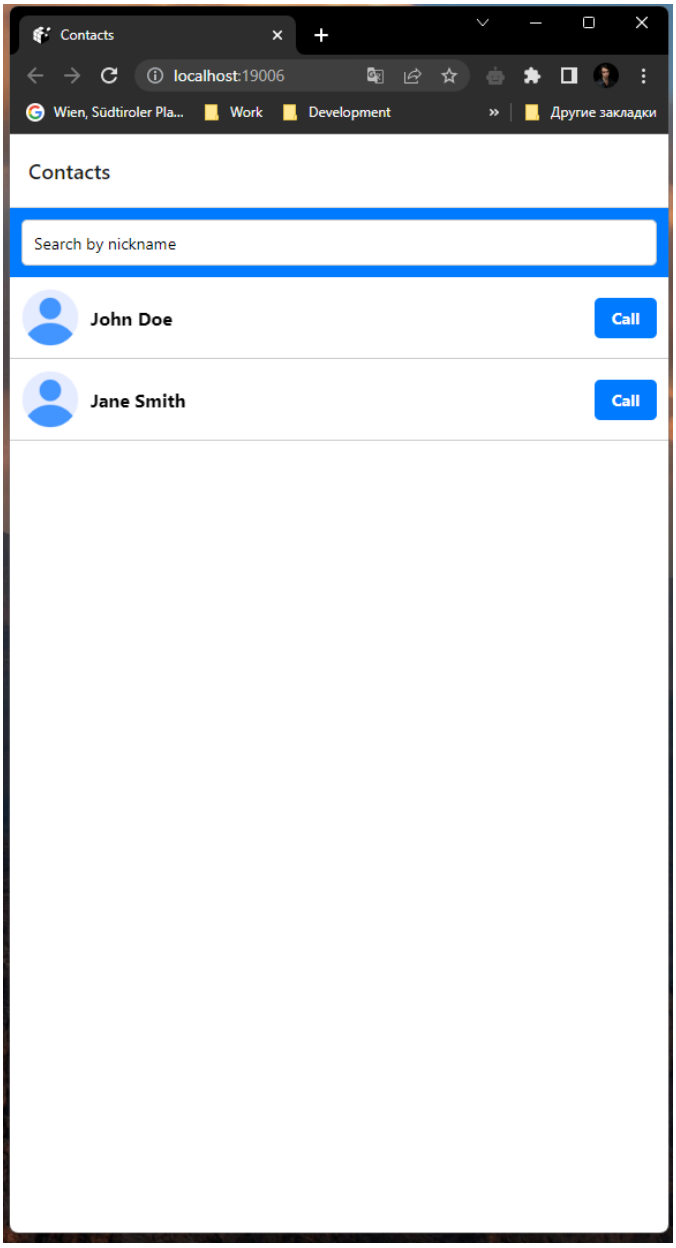


Рисунок 3.7 – Екран контактів

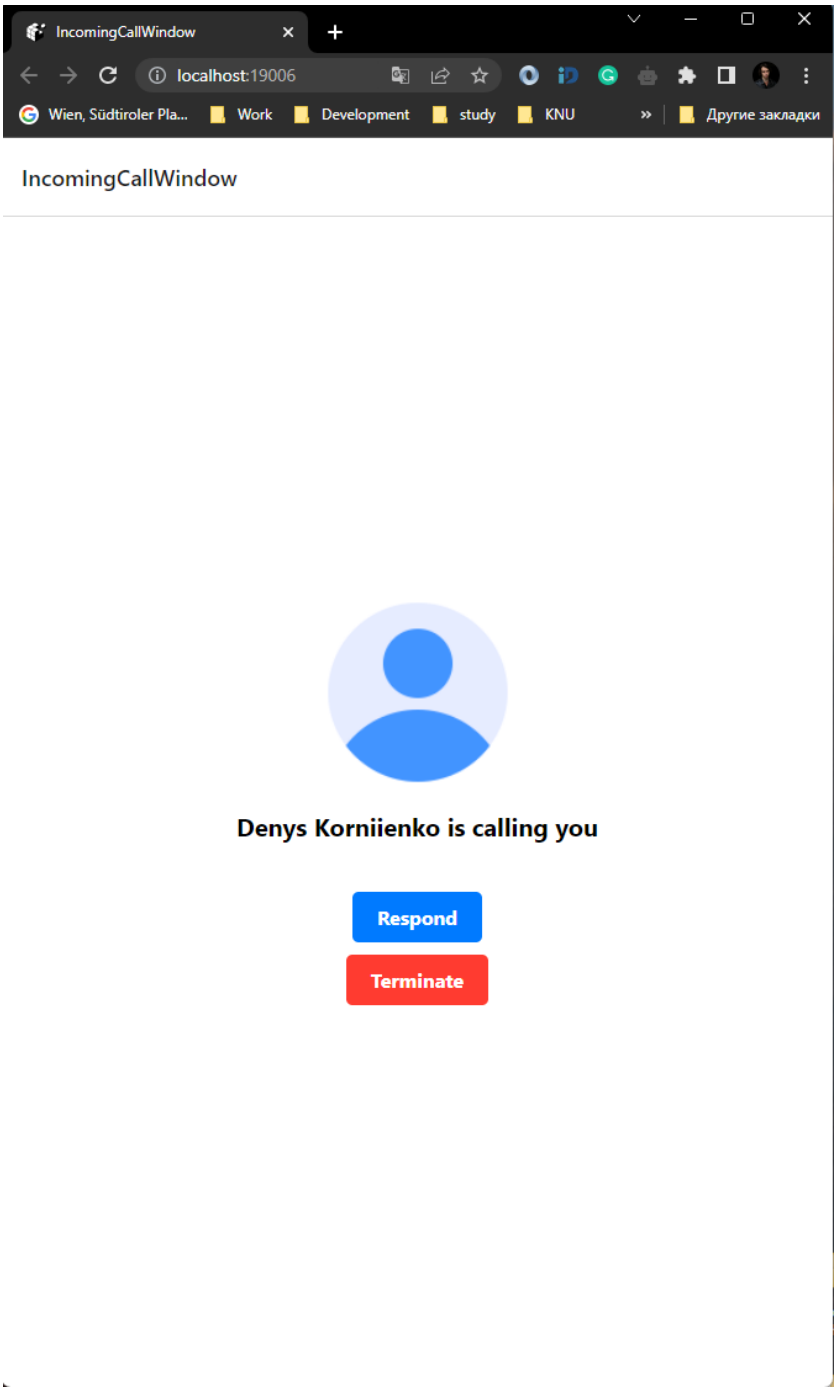


Рисунок 3.8 – Вікно вхідного дзвінка

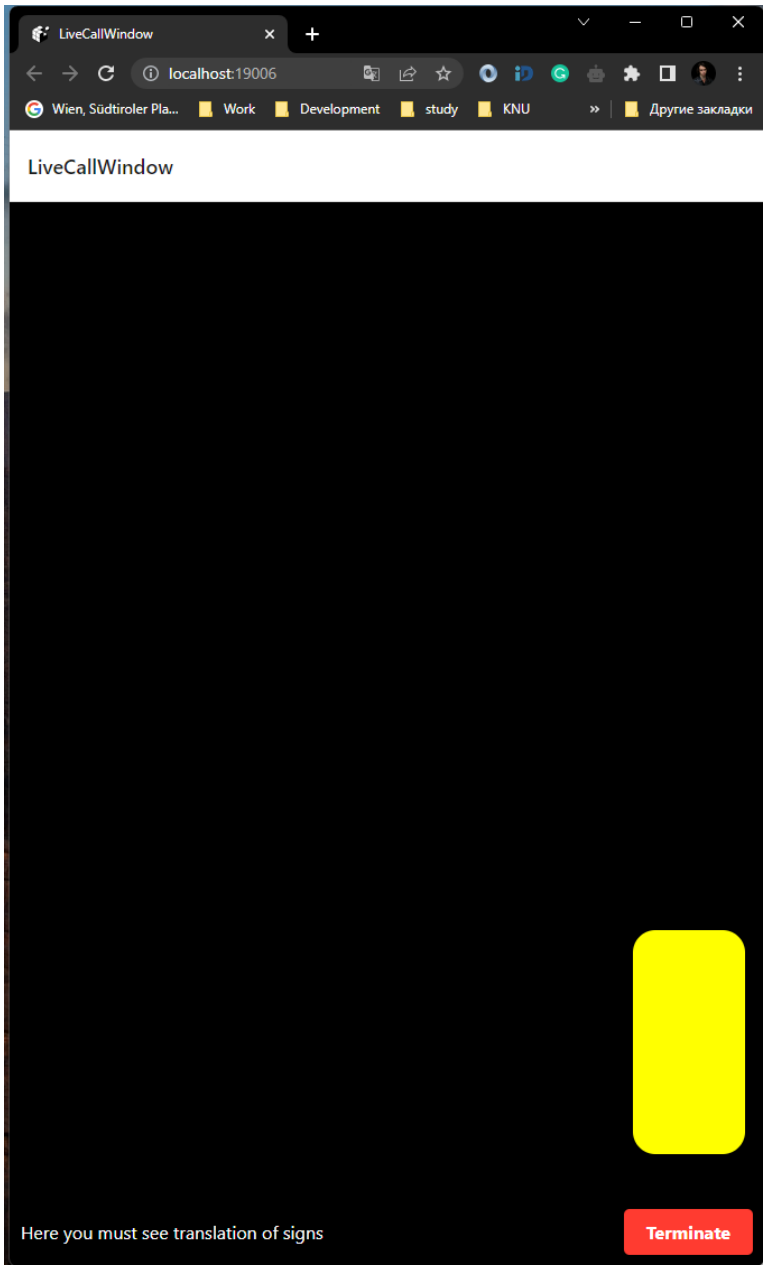


Рисунок 3.9 – Вікно дзвінка

Далі було створено WebAPI компонент на плафтормі dotNet для організації авторизації, створення користувача, викачці контактів а також для встановлення з’єднання.

3.2 Організація тестування мобільного додатка відеозв’язку

Для тестування мобільного додатка було збілдено Firebarrier.apk файл для встановлення програми. Далі булі провдені такі кроки:

1. Користувач А (автор роботи) встановив та відкрив додаток на своєму мобільному девайсі.
2. Користувач Б (Максим Повжнюк) встановив та відкрив додаток на своєму мобільному девайсі.
3. Користувач А і Б авторизувалися через голоивний екран додатка.
4. Користувач А знайшов користувача Б в контактах і зателефанував користувачу Б натиснувши кнопку “Call”.
5. В користувача Б з’явилось вікно вхідноного виклику. Користувач натиснув кнопку “Respond”.
6. З’єднання між користувачем А та Б було встановлено. Дзвінок тривав 5 хв.
7. Користувач А завершив дзвінок натиснувши кнопку “Terminate”.

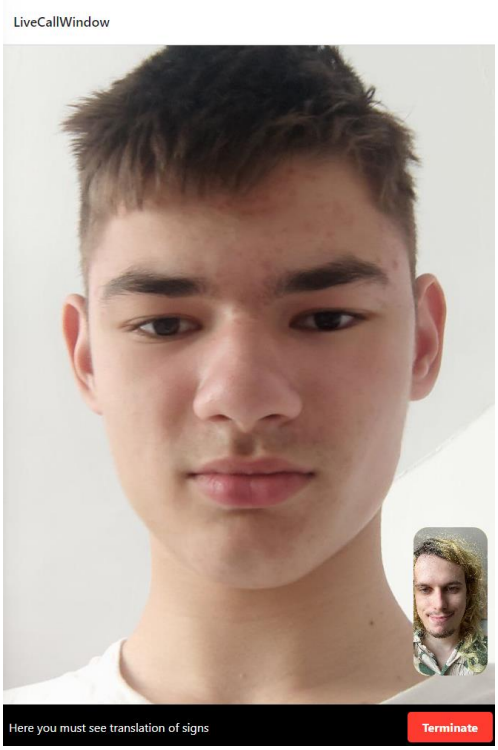


Рисунок 3.10 – Скріншот роботи мобільного додатку під час виклику

3.3 Реалізація моделі нейроної мережі для розпізнавання образів

Для побудови моделі, було розроблено модуль навчання нейроної мережі на мові програмування Python і на основі бібліотеки TensorFlow та репозиторія “tensorflow/models” [21].

На самому початку було підготовлено власний дата сет із зображеннями жестів. Датасет включає до 800 зображень зроблених власноруч.

Рисунок 3.11 – Зображення букви “А”

Рисунок 3.12 – Зображення букви “Я”

Зображення датасету було зроблені для 29 літер (класів) української жестової абетки включаючи такі літери: А, Б, В, Г, Д, Е, Ж, З, И, І, К, Л, М, Н, О, П, Р, С, Т, У, Ф, Х, Ц, Ч, Ш, Щ, Ь, Ю, Я [2].

Було налаштовано конфігурацію (Додаток 1) для навчання моделі. Всього дана модель має 3 основних шарів:

1. Вхідний шар (Input Layer): Вхідне зображення розміром 320x320 пікселів.
2. SSD MobileNet v2 FPN (Feature Pyramid Network): Цей шар використовує архітектуру SSD MobileNet v2 FPN і має декілька згорткових шарів.
3. Предиктор обмежувальних рамок (Box Predictor): Цей шар використовує згорткові шари для передбачення параметрів обмежувальних рамок. Має 128 нейронів у глибині.

За допомогою інструмета LabelImg було згенеровано “лейбли” для

зображень.

Далі для навчання було згенеровано файли tensorflow (“record-и”) які містять в собі бінарні дані зображень та їх опис, за допомогою утиліти learn_data_prepation_util (Додаток 2).

Запущено утиліту навчання learn_util і подано їй на вхід “record-и”. Навчання відбувалося на основі NVIDIA GPU Computing Toolkit (пакет розробки засобів для високонапружених обчислень на відеокартах від NVIDIA за допомогою технології CUDA) [22]. Процес начання тривав близько 2 годин.

Після навчання нейроної мережі було отримано модель для розпізнавання жестів.

3.4 Тестування моделі нейромережі для розпізнавання жестів

Кроки тестування:

1. Запуск утиліти model_test (Додаток 3)
2. Відкриття вікна зображення веб-камери в режимі реального часу.
3. Тестувальник здійснює жести абетки.
4. Жести, що було розпізнато обводяться рамкою. Рамка містить текст літери, які були розізнані та відсоткове значення точності розпізнавання.

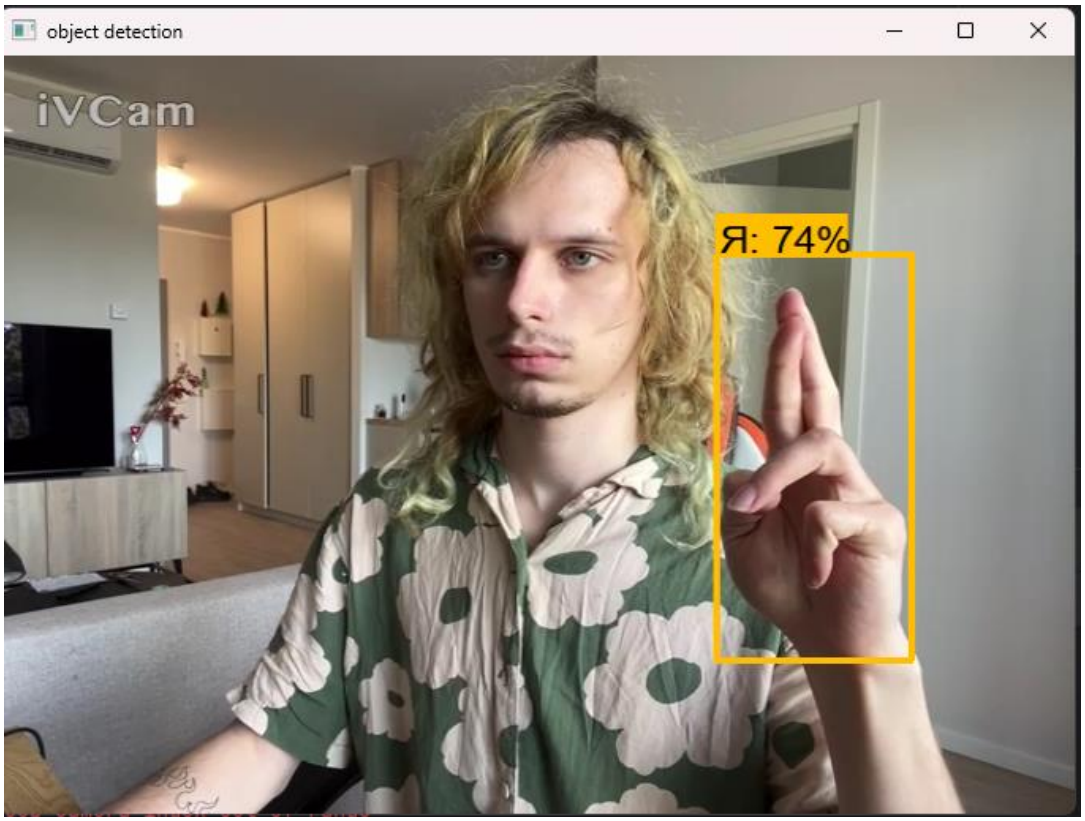


Рисунок 3.13 – Скріншот роботи утиліти model_test і розпізнавання жестів в реальному часі, розпізнано літеру Я

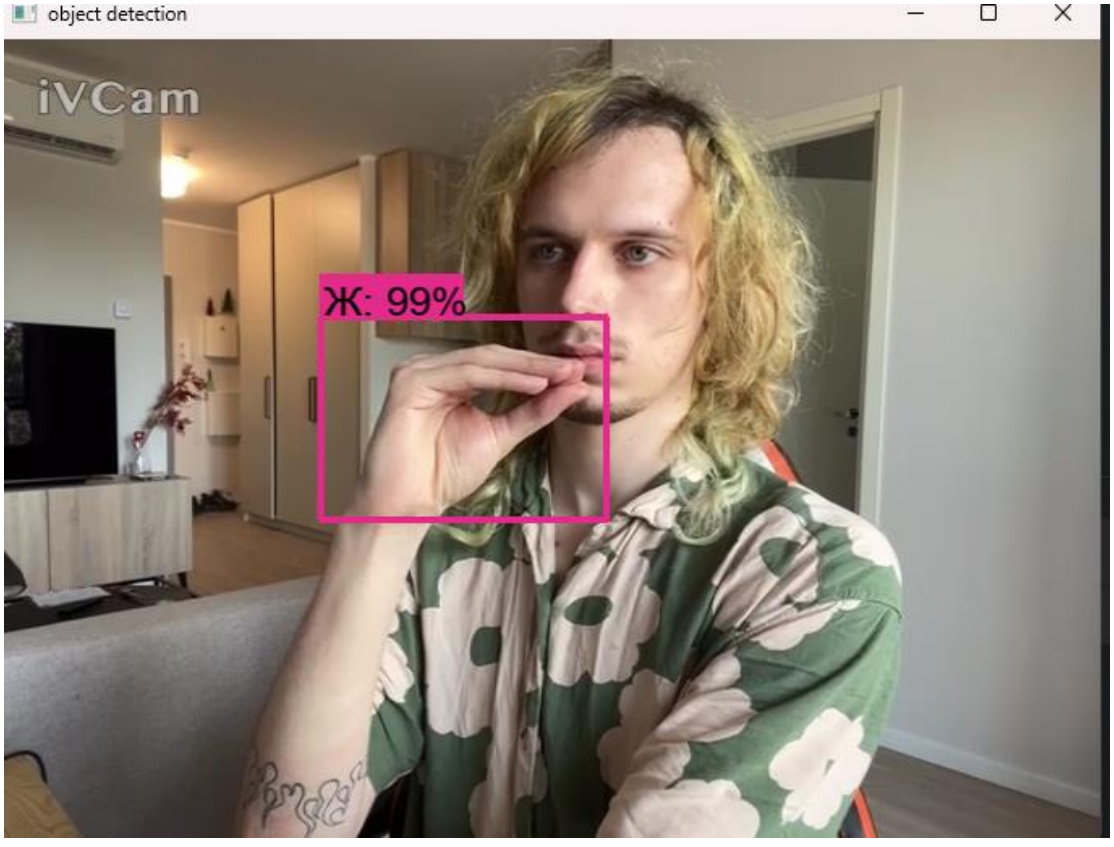


Рисунок 3.14 – Скріншот роботи утиліти model_test і розпізнавання жестів в реальному часі, розпізнано літеру Ж

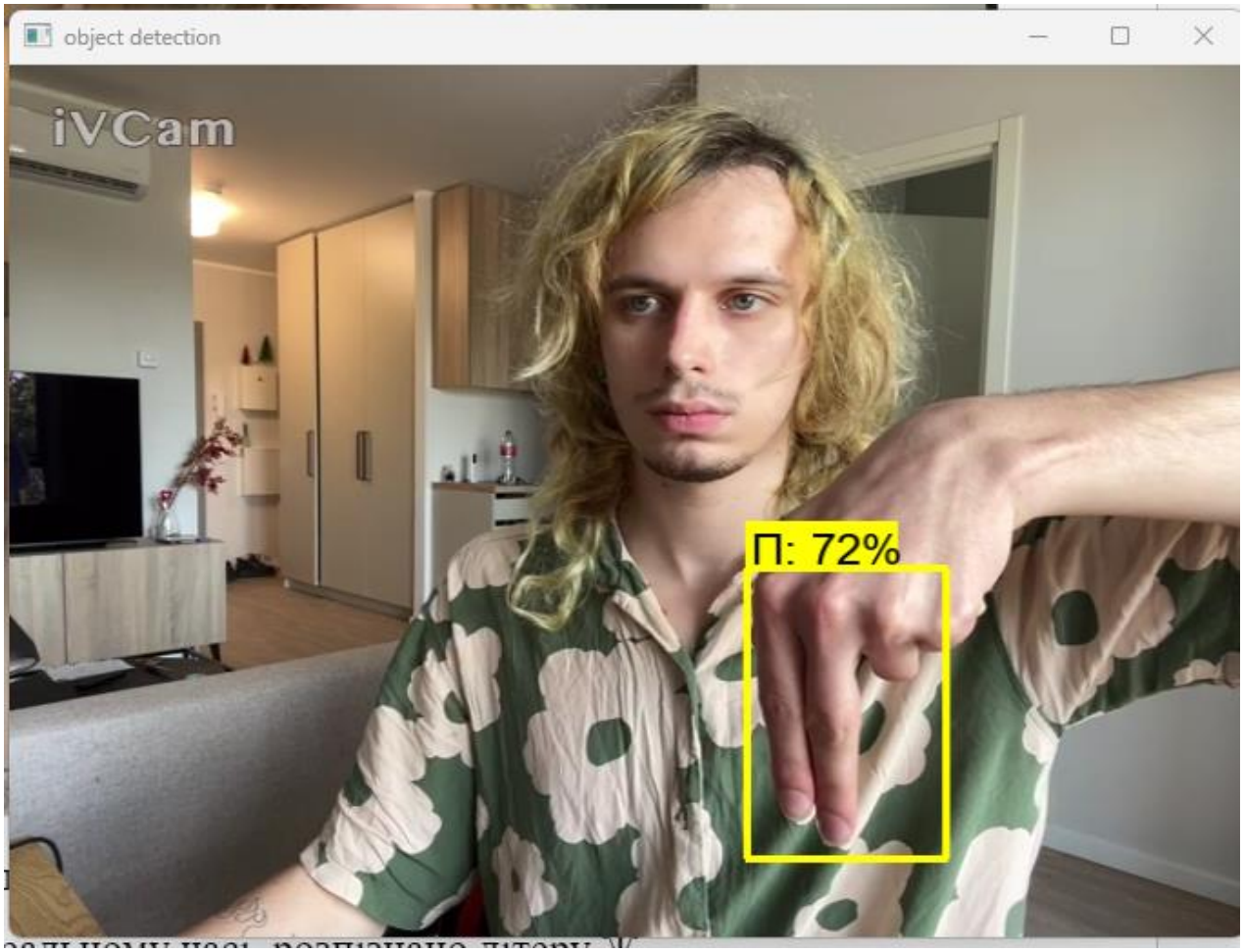


Рисунок 3.15 – Скріншот роботи утиліти model_test і розпізнавання жестів в реальному часі, розпізнано літеру П

Отже, під час роботи над цим розділом було розроблено концепти та сам мобільний додаток, WebAPI модуль, модулі для навчання і розпізнавання жестів.

ВИСНОВКИ

В результаті виконання цієї кваліфікаційної роботи було досліджено особливості задачі розпізнавання жестів та відеозв'язку через інтернет. Для реалізації моделі розпізнавання жестів було визначено доречним використання

алгоритму навчання SSD моделі. Для цієї роботи було створено власний датасет зображень жестів літер української мови. Побудовано архітектуру основних компонентів системи, діаграму діяльності, дерево функцій і діаграму послідовності. Для розробки мобільного додатка було сформовано концепт-зображення його інтерфейса. Мобільний додаток розроблено на основі ReactNative. Було обрано протокол відеозв'язку WebRTC через його відносно просту інтеграцію, розширену документацію [11] та кросплатформеність. Було побудовано бекенд-модуль WebAPI для роботи із БД, авторизації і створення викликів. Були наведені результати роботи мобільного додатка в якості скріншотів.

Також, створено і навчено модель розпізнавання образів за допомогою вищезгаденого датасету та утиліт і моделів написаних на мові Python (розглянуто в 3-у розділі). Побудовано невеликий додаток для розпізнавання жестів з веб-камери в режимі реального часу.

Нажаль, також потрібно згадати негативний результат цієї роботи. Не вдалося об'єднати вищезгадені додатки розпізнавання жестів з веб-камери в режимі реального часу та мобільний додаток-месенджер. Одним з рішень інтеграції tensorflow моделі було встановити пакет TensorflowJs поверх мобільного додатка, що був написаний на React. TensorflowJs може використовувати і завантажувати у пам'ять мобільних присроїв моделі у форматі Keras.h5. Отримана модель розпізнавання образів була у форматі .pb який не є сумісним із TensorflowJs. Для вирішення цієї проблеми існує два варіанти:

1. Побудувати нову модель із правильним форматом слідуючи наступним інструкціям [23].
2. Створити новий API сервіс на основі Flask пакету [24] і використовувати існуючу модель, як у тестовому модулі model_test (Додаток 3) розпізнаючі кадри зображень, які будуть натходити від мобільних додатків чере POST метод протоколу HTTP під час виклику та видавати результати назад клієнтам.

В цілому додаток описаний в цій роботі має високу практичну цінність, адже, він поліпшить комунікацію на відстані між людьми що мають вади слуху

та здоровими людьми. Майбутній готовий застосунок можна застосувати у різних сферах життя: подібну систему можна інтегрувати в органи швидкого реагування (поліцію, швидку допомогу, тощо), або ця система може бути застосована для організації бізнес конференцій, у яких зможуть брати участь глухонімі люди.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Жестова мова як засіб комунікації. [Електронний ресурс] – Електр. дані.– Режим доступу : https://ukrbukva.net/103193-Zhestovyiy-yazyk-kak-sredstvo-kommunikacii.html
2. Українська жестова абетка. [Електронний ресурс] – Електр. дані.– Режим доступу : https://lu-ua.com.ua/userfiles/image/catalog/thumbs/photoimg_1189966667_big_photo.jpg
3. Жест “Привіт”. [Електронний ресурс] – Електр. дані.– Режим доступу : https://media.spreadthesign.com/video/mp4/31/110551.mp4
4. Жест “Вибач”. [Електронний ресурс] – Електр. дані.– Режим доступу : https://media.spreadthesign.com/video/mp4/30/102370.mp4
5. Жест “Так”. [Електронний ресурс] – Електр. дані.– Режим доступу : https://media.spreadthesign.com/video/mp4/13/485639.mp4
6. Інтерфейс додатку Telegram. [Електронний ресурс] – Електр. дані.– Режим доступу : https://telegram.org/file/811140117/1/U8DtuwqizBc/f31a8bb95bdf993cea
7. WhatsApp [Електронний ресурс] – Електр. дані.– Режим доступу : https://www.91-cdn.com/hub/wp-content/uploads/2022/11/WhatsApp-Message-Yourself.jpg
8. Zoom [Електронний ресурс] – Електр. дані.– Режим доступу : https://blog.zoom.us/zoom-rolling-out-end-to-end-encryption-offering

9. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — алгоритми розпізнавання образів – [Електронний ресурс] – Електр. дані.– Режим доступу : https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
10. Як працює одноразовий детектор (SSD)? [Електронний ресурс] – Електр. дані.– Режим доступу : https://developers.arcgis.com/python/guide/how-ssd-works/
11. Ракеш Арора : Voice over IP : Protocols and Standards – Електр. дані. – Режим доступу : https://www.researchgate.net/publication/238430098_Voice_over_IP_Protocols_and_Standards
12. VOIP/PIP PHONES [Електронний ресурс] – Електр. дані.– Режим доступу : https://www.netcomplete.net/voippip-phones/
13. WebRTC API [Електронний ресурс] – Електр. дані.– Режим доступу : https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API
14. Що таке WebRTC? [Електронний ресурс] – Електр. дані.– Режим доступу : https://trueconf.com/webrtc.html
15. TensorFlow [Електронний ресурс] – Електр. дані.– Режим доступу : https://www.tensorflow.org/
16. labelImg [Електронний ресурс] – Електр. дані.– Режим доступу : https://github.com/heartexlabs/labelImg
17. Підтримка браузера WebRTC на комп'ютері та мобільному пристрої [Електронний ресурс] – Електр. дані.– Режим доступу : https://bloggeek.me/webrtc-browser-support/
18. Почніть роботу з WebRTC [Електронний ресурс] – Електр. дані.– Режим доступу : https://web.dev/webrtc-basics/
19. Speech-to-Text [Електронний ресурс] – Електр. дані.– Режим доступу : https://cloud.google.com/speech-to-text
20. NodeJS [Електронний ресурс] – Електр. дані.– Режим доступу : https://nodejs.org/en/download
21. TensorFlow models [Електронний ресурс] – Електр. дані.– Режим доступу : https://github.com/tensorflow/models/tree/master

22. CUDA Toolkit 10.2 [Електронний ресурс] – Електр. дані.– Режим доступу : https://developer.nvidia.com/cuda-10.2-download-archive

23. Кастомне виявлення об'єктів у браузері за допомогою TensorFlow.js [Електронний ресурс] – Електр. дані.– Режим доступу : https://blog.tensorflow.org/2021/01/custom-object-detection-in-browser.html

24. Flask [Електронний ресурс] – Електр. дані.– Режим доступу : *https://flask.palletsprojects.com/en/2.3.x/*

ДОДАТКИ

Додаток 1. Конфігурація SSD моделі

```
model {
 ssd {
  num_classes: 29
  image_resizer {
   fixed_shape_resizer {
    height: 320
    width: 320
   }
  }
  feature_extractor {
   type: "ssd_mobilenet_v2_fpn_keras"
   depth_multiplier: 1.0
   min_depth: 16
   conv_hyperparams {
    regularizer {
     l2_regularizer {
      weight: 4e-05
     }
    }
    initializer {
     random_normal_initializer {
      mean: 0.0
      stddev: 0.01
     }
    }
    activation: RELU_6
    batch_norm {
     decay: 0.997
     scale: true
     epsilon: 0.001
    }
   }
   use_depthwise: true
   override_base_feature_extractor_hyperparams: true
```

```
  fpn {
    min_level: 3
    max_level: 7
    additional_layer_depth: 128
  }
}
box_coder {
  faster_rcnn_box_coder {
    y_scale: 10.0
    x_scale: 10.0
    height_scale: 5.0
    width_scale: 5.0
  }
}
matcher {
  argmax_matcher {
    matched_threshold: 0.5
    unmatched_threshold: 0.5
    ignore_thresholds: false
    negatives_lower_than_unmatched: true
    force_match_for_each_row: true
    use_matmul_gather: true
  }
}
similarity_calculator {
  iou_similarity {
  }
}
box_predictor {
  weight_shared_convolutional_box_predictor {
    conv_hyperparams {
      regularizer {
        l2_regularizer {
          weight: 4e-05
        }
      }
      initializer {
        random_normal_initializer {
          mean: 0.0
          stddev: 0.01
        }
      }
      activation: RELU_6
      batch_norm {
        decay: 0.997
        scale: true
        epsilon: 0.001
      }
    }
    depth: 128
    num_layers_before_predictor: 4
    kernel_size: 3
    class_prediction_bias_init: -4.6
    share_prediction_tower: true
    use_depthwise: true
  }
}
anchor_generator {
  multiscale_anchor_generator {
    min_level: 3
```

```
      max_level: 7
      anchor_scale: 4.0
      aspect_ratios: 1.0
      aspect_ratios: 2.0
      aspect_ratios: 0.5
      scales_per_octave: 2
    }
  }
  post_processing {
    batch_non_max_suppression {
      score_threshold: 1e-08
      iou_threshold: 0.6
      max_detections_per_class: 100
      max_total_detections: 100
      use_static_shapes: false
    }
    score_converter: SIGMOID
  }
  normalize_loss_by_num_matches: true
  loss {
    localization_loss {
      weighted_smooth_l1 {
      }
    }
    classification_loss {
      weighted_sigmoid_focal {
        gamma: 2.0
        alpha: 0.25
      }
    }
    classification_weight: 1.0
    localization_weight: 1.0
  }
  encode_background_as_zeros: true
  normalize_loc_loss_by_codesize: true
  inplace_batchnorm_update: true
  freeze_batchnorm: false
 }
}
train_config {
 batch_size: 4
 data_augmentation_options {
  random_horizontal_flip {
  }
 }
 data_augmentation_options {
  random_crop_image {
    min_object_covered: 0.0
    min_aspect_ratio: 0.75
    max_aspect_ratio: 3.0
    min_area: 0.75
    max_area: 1.0
    overlap_thresh: 0.0
  }
 }
 sync_replicas: true
 optimizer {
  momentum_optimizer {
    learning_rate {
      cosine_decay_learning_rate {
```

```
        learning_rate_base: 0.08
        total_steps: 50000
        warmup_learning_rate: 0.026666
        warmup_steps: 1000
      }
    }
    momentum_optimizer_value: 0.9
  }
  use_moving_average: false
 }
 fine_tune_checkpoint: "D:\\Study\\Taras Shevchenko\\Diplom\\sign_language_service/pre-trained-
models/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-0"
 num_steps: 50000
 startup_delay_steps: 0.0
 replicas_to_aggregate: 8
 max_number_of_boxes: 100
 unpad_groundtruth_tensors: false
 fine_tune_checkpoint_type: "detection"
 fine_tune_checkpoint_version: V2
}
train_input_reader {
 label_map_path: "D:\\Study\\Taras
Shevchenko\\Diplom\\sign_language_service/output/annotations/label_map.pbtxt"
 tf_record_input_reader {
   input_path: "D:\\Study\\Taras
Shevchenko\\Diplom\\sign_language_service/output/annotations/train.record"
 }
}
eval_config {
 metrics_set: "coco_detection_metrics"
 use_moving_averages: false
}
eval_input_reader {
 label_map_path: "D:\\Study\\Taras
Shevchenko\\Diplom\\sign_language_service/output/annotations/label_map.pbtxt"
 shuffle: false
 num_epochs: 1
 tf_record_input_reader {
   input_path: "D:\\Study\\Taras
Shevchenko\\Diplom\\sign_language_service/output/annotations/test.record"
 }
}
```

Додаток 2. Утиліта навчання мережі learn_data_prepation_util

```
import os
import shutil
import codecs
from config import *

import scripts.generate_tfrecord as gtf

# label mask
labels = [{'name':'А', 'id':1},
        {'name':'Б', 'id':2},
        {'name':'В', 'id':3},
        {'name':'Г', 'id':4},
        {'name':'Д', 'id':5},
        {'name':'Е', 'id':6},
        {'name':'Є', 'id':7},
```

```
        {'name':'Ж', 'id':8},
        {'name':'З', 'id':9},
        {'name':'И', 'id':10},
        {'name':'І', 'id':11},
        {'name':'К', 'id':12},
        {'name':'Л', 'id':13},
        {'name':'М', 'id':14},
        {'name':'Н', 'id':15},
        {'name':'О', 'id':16},
        {'name':'П', 'id':17},
        {'name':'Р', 'id':18},
        {'name':'С', 'id':19},
        {'name':'Т', 'id':20},
        {'name':'У', 'id':21},
        {'name':'Ф', 'id':22},
        {'name':'Х', 'id':23},
        {'name':'Ц', 'id':24},
        {'name':'Ч', 'id':25},
        {'name':'Ш', 'id':26},
        {'name':'Ь', 'id':27},
        {'name':'Ю', 'id':28},
        {'name':'Я', 'id':29}
        ]

with codecs.open(ANNOTATION_PATH + '/label_map.pbtxt', 'w', 'utf-8') as f:
  for label in labels:
      f.write('item { \n')
      f.write('\tname:\'{}\'\n'.format(label['name']))
      f.write('\tid:{}\n'.format(label['id']))
      f.write('}\n')

gtf.main(xml_dir=IMAGE_PATH + '/train', labels_path=ANNOTATION_PATH + '/label_map.pbtxt',
output_path=ANNOTATION_PATH + '/train.record')
gtf.main(xml_dir=IMAGE_PATH + '/test', labels_path=ANNOTATION_PATH + '/label_map.pbtxt',
output_path=ANNOTATION_PATH + '/test.record')
```

Додаток 3. Модуль розпізнавання жестів в режимі реального часу model_test

```
import cv2
import numpy as np
import os
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from config import *

# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(CONFIG_PATH)
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(CHECKPOINT_PATH, 'ckpt-101')).expect_partial()

@tf.function
```

```
def detect_fn(image):
  image, shapes = detection_model.preprocess(image)
  prediction_dict = detection_model.predict(image, shapes)
  print(prediction_dict)
  detections = detection_model.postprocess(prediction_dict, shapes)
  return detections

category_index =
label_map_util.create_category_index_from_labelmap(ANNOTATION_PATH+'/label_map.pbtxt')
# Setup capture
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
#cap.release()

camNums = []
print(camNums)
for i in range(100):
  if cv2.VideoCapture(i).isOpened():
    camNums.append(i)

print(camNums)

while True:
  ret, frame = cv2.VideoCapture(0).read()
  image_np = np.array(frame)

  input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
  detections = detect_fn(input_tensor)

  num_detections = int(detections.pop('num_detections'))
  detections = {key: value[0, :num_detections].numpy()
          for key, value in detections.items()}
  detections['num_detections'] = num_detections

  # detection_classes should be ints.
  detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

  label_id_offset = 1
  image_np_with_detections = image_np.copy()

  viz_utils.visualize_boxes_and_labels_on_image_array(
    image_np_with_detections,
    detections['detection_boxes'],
    detections['detection_classes'] + label_id_offset,
    detections['detection_scores'],
    category_index,
    use_normalized_coordinates=True,
    max_boxes_to_draw=5,
    min_score_thresh=.5,
    agnostic_mode=False)

  cv2.imshow('object detection', cv2.resize(image_np_with_detections, (680, 480)))

  if cv2.waitKey(1) & 0xFF == ord('q'):
    cap.release()
    break
```